



UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA

# Traducción dirigida por la sintaxis (TDS - SDT)

**Felipe Restrepo Calle**

[ferestrepoca@unal.edu.co](mailto:ferestrepoca@unal.edu.co)

Lenguajes de Programación  
Departamento de Ingeniería de Sistemas e Industrial  
Facultad de Ingeniería  
Universidad Nacional de Colombia  
Sede Bogotá

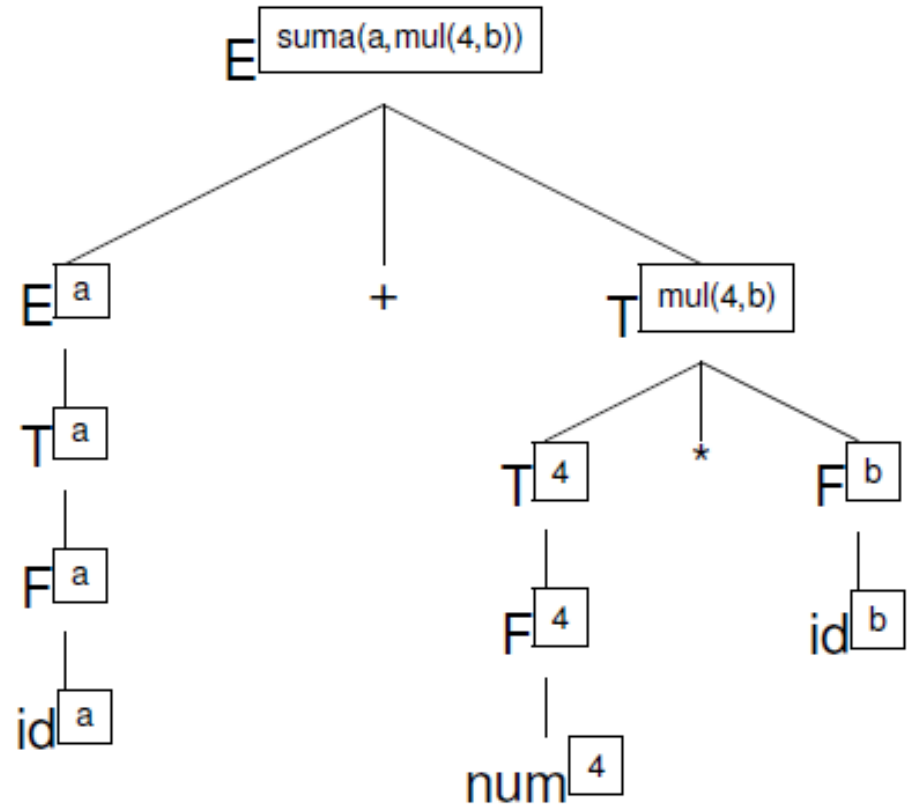
1. Gramáticas de atributos
2. Definición Dirigida por Sintaxis – DDS
3. Esquemas de Traducción Dirigidos por Sintaxis – ETDS
4. Ejercicios



## Ejemplo 1 de traducción

$E \rightarrow E + T$   
 $E \rightarrow T$   
 $T \rightarrow T * F$   
 $T \rightarrow F$   
 $F \rightarrow \text{id}$   
 $F \rightarrow \text{num}$   
 $F \rightarrow \text{pari } E \text{ pard}$

$a + 4 * b \Rightarrow \text{suma}(a, \text{mul}(4, b))$





## Ejemplo 1 de traducción

### Gramática de atributos

REGLA	ACCIÓN SEMÁNTICA
$E \rightarrow E + T$	$E.trad := "suma("    E_1.trad    ", "    T.trad    ")"$
$E \rightarrow T$	$E.trad := T.trad$
$T \rightarrow T * F$	$T.trad := "mul("    T_1.trad    ", "    F.trad    ")"$
$T \rightarrow F$	$T.trad := F.trad$
$F \rightarrow id$	$F.trad := id.lexema$
$F \rightarrow num$	$F.trad := num.lexema$
$F \rightarrow pari E pard$	$F.trad := E.trad$

**IMPORTANTE:** no es posible acceder a atributos de símbolos que no estén en la regla

## Implementación del traductor

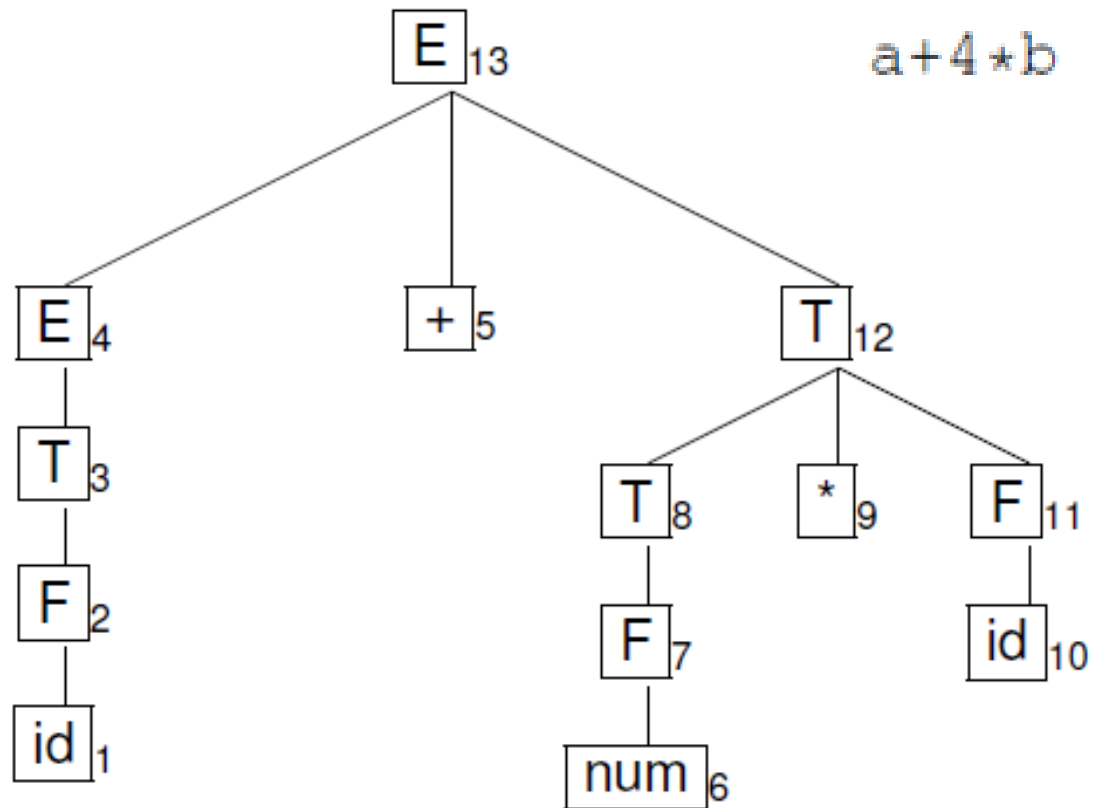
Existen dos posibilidades:

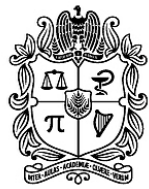
1. El analizador sintáctico construye el árbol (decorado con los atributos de los terminales), y en una segunda pasada se recorre el árbol calculando los atributos que falten hasta completar la traducción (**traducción de dos o más pasadas**)
2. El analizador sintáctico no construye explícitamente el árbol (aunque hace un recorrido virtual por el árbol), y va calculando todos los atributos a la vez que va recorriendo el árbol (**traducción de una sola pasada**)



## Recorrido virtual del árbol realizado por el analizador sintáctico

$E \rightarrow E + T$   
 $E \rightarrow T$   
 $T \rightarrow T * F$   
 $T \rightarrow F$   
 $F \rightarrow \text{id}$   
 $F \rightarrow \text{num}$   
 $F \rightarrow \text{pari } E \text{ pard}$

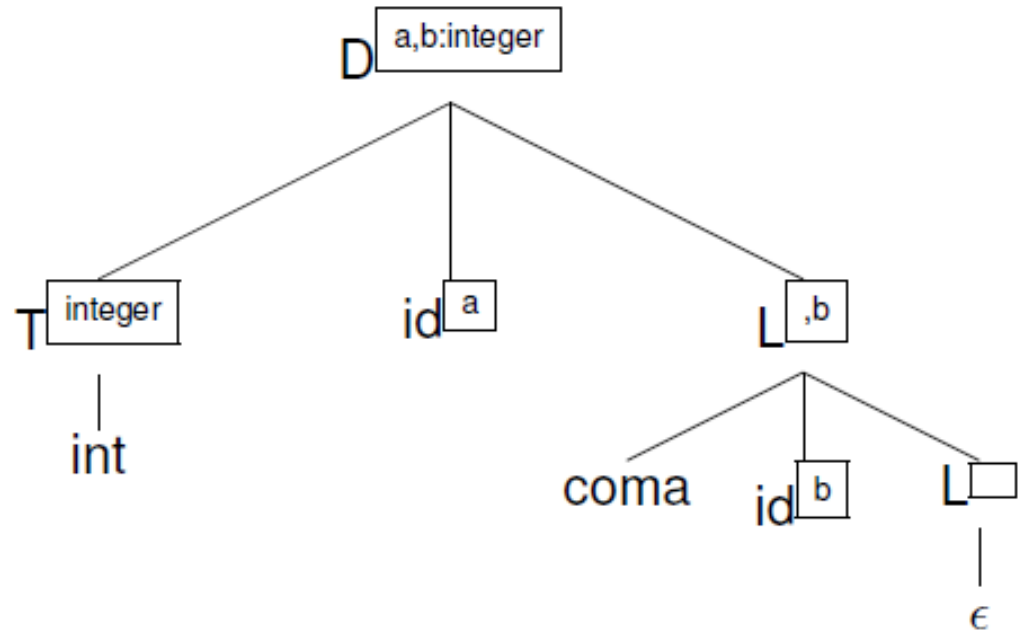


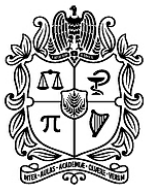


## Ejemplo 2 de traducción

$D \rightarrow T \text{ id } L$   
 $T \rightarrow \text{float}$   
 $T \rightarrow \text{int}$   
 $L \rightarrow \text{coma id } L$   
 $L \rightarrow \epsilon$

`int a,b`  $\Rightarrow$  `a,b:integer`





## Ejemplo 2 de traducción

### Gramática de atributos

REGLA	ACCIÓN SEMÁNTICA
$D \rightarrow T \text{ id } L$	$D.trad := \text{id.lexema}    L.trad    " : "    T.trad$
$T \rightarrow \text{float}$	$T.trad := \text{"real"}$
$T \rightarrow \text{int}$	$T.trad := \text{"integer"}$
$L \rightarrow \text{coma id } L$	$L.trad := ", "    \text{id.lexema}    L_1.trad$
$L \rightarrow \epsilon$	$L.trad := ""$

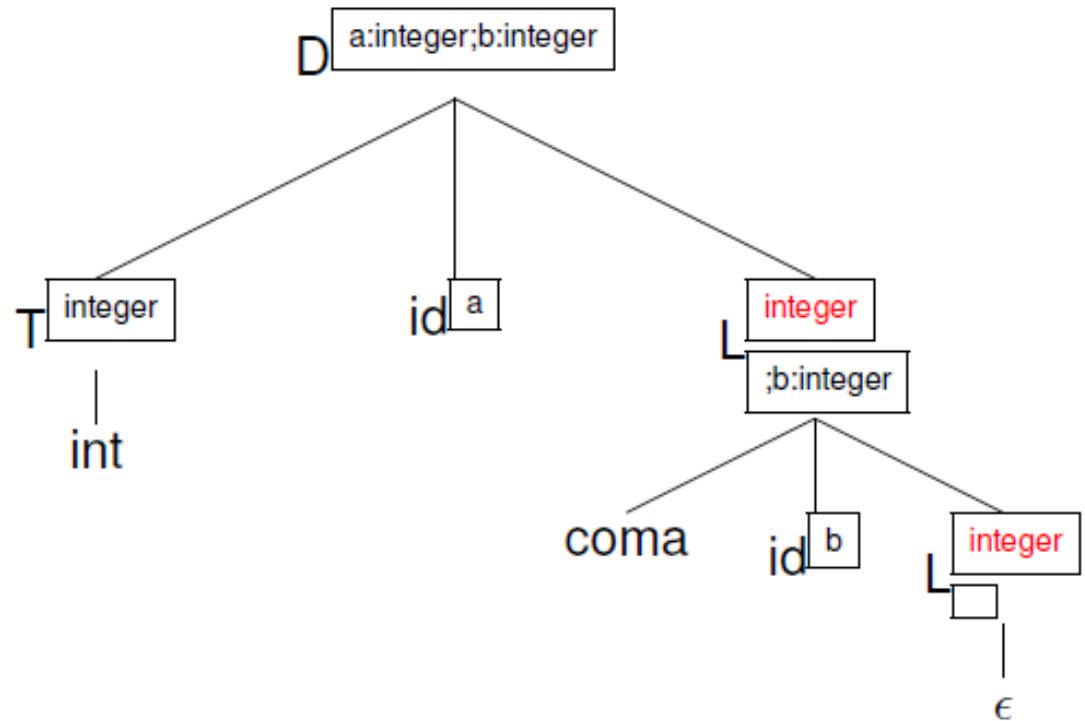




## Ejemplo 3 de traducción

$D \rightarrow T \text{ id } L$   
 $T \rightarrow \text{float}$   
 $T \rightarrow \text{int}$   
 $L \rightarrow \text{coma id } L$   
 $L \rightarrow \epsilon$

`int a,b`  $\Rightarrow$   
`a:integer;b:integer`





## Ejemplo 3 de traducción

### Gramática de atributos

REGLA	ACCIÓN SEMÁNTICA
$D \rightarrow T \text{ id } L$	$L.th := T.trad;$ $D.trad := \text{id.lexema}    " : "    T.trad    L.trad$
$T \rightarrow \text{float}$	$T.trad := \text{"real"}$
$T \rightarrow \text{int}$	$T.trad := \text{"integer"}$
$L \rightarrow \text{coma id } L$	$L_1.th := L.th;$ $L.trad := " ; "    \text{id.lexema}    " : "    L.th    L_1.trad$
$L \rightarrow \epsilon$	$L.trad := ""$

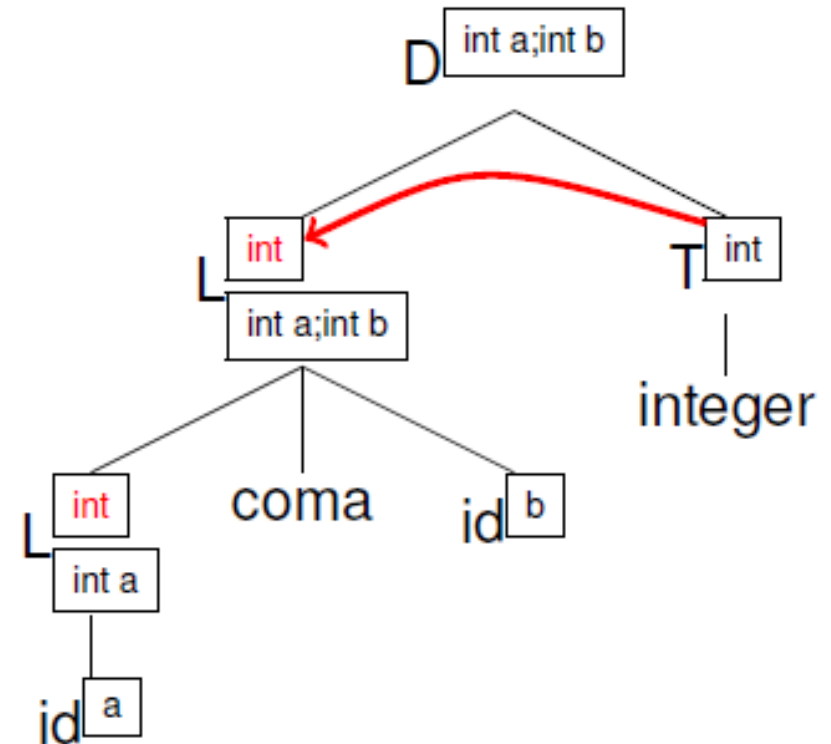
- Algunos no terminales tienen **más de un atributo**
- Algunos atributos se calculan a partir de atributos de hermanos o padres en el árbol. Estos atributos se llaman **atributos heredados** (los demás atributos se llaman **sintetizados**).



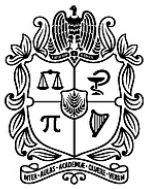
## Ejemplo 4 de traducción

$D \rightarrow L T$   
 $T \rightarrow \text{real}$   
 $T \rightarrow \text{integer}$   
 $L \rightarrow L \text{ coma id}$   
 $L \rightarrow \text{id}$

`a, b integer`  $\Rightarrow$  `int a; int b`



**IMPORTANTE:** hay herencia de derecha a izquierda en el árbol



## Ejemplo 4 de traducción

### Gramática de atributos

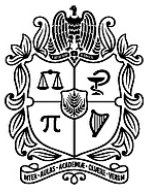
REGLA	ACCIÓN SEMÁNTICA
$D \rightarrow L T$	$L.th := T.trad; D.trad := L.trad$
$T \rightarrow \text{real}$	$T.trad := \text{"double"}$
$T \rightarrow \text{integer}$	$T.trad := \text{"int"}$
$L \rightarrow L \text{ coma } id$	$L_1.th := L.th;$ $L.trad := L_1.trad    \text{" ; " }    L.th    \text{" " }    id.lexema$
$L \rightarrow id$	$L.trad := L.th    \text{" " }    id.lexema$

- La herencia de derecha (T) a izquierda (L) implica que cuando se está analizando/traduciendo L todavía no se conoce la traducción de T
- Por tanto, no es posible traducir a la vez que se analiza, **no es posible la traducción en una pasada.**

## Gramática de atributos por la izquierda

Una gramática de atributos por la izquierda (***l-attributed grammar***):

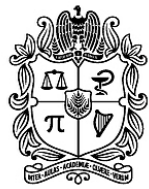
- Solamente presenta herencia de padres a hijos o de izquierda a derecha en el árbol, pero no de derecha a izquierda.
- Permite evaluar los atributos a la vez que se hace el análisis sintáctico (**en una sola pasada**).
- Cuando tiene herencia de derecha a izquierda en el árbol, el analizador sintáctico debe construir un árbol decorado con atributos y, en una segunda pasada, evaluar los atributos que no se hayan podido evaluar en la primera pasada.



## Definición Dirigida por la Sintaxis – DDS

### Tabla de símbolos

- Cuando el compilador procesa las declaraciones, tiene que añadir las variables declaradas a la tabla de símbolos (comprobando que no se declara dos veces el mismo identificador en el mismo ámbito).
- Cuando aparece un identificador en el código (instrucciones, expresiones, etc.), el compilador debe buscarlo en la tabla de símbolos, y obtener sus datos: tipo, dirección, etc.



## Ejemplo 5 de traducción

- No genera traducción, debe almacenar los símbolos en la tabla.
- Cuando una gramática de atributos tiene acciones que no calculan atributos, se denomina **Definición dirigida por la sintaxis (DDS)**:

`int a,b,c`

NOMBRE	TIPO	DIRECCIÓN
a	ENTERO	0
b	ENTERO	2
c	ENTERO	4

### DEFINICIÓN DIRIGIDA POR LA SINTAXIS

REGLA	ACCIÓN SEMÁNTICA
$D \rightarrow T \text{ id } L$	$L.th := T.tipo; \text{anadirTS}(\text{id.lexema}, T.tipo)$
$T \rightarrow \text{float}$	$T.tipo := REAL$
$T \rightarrow \text{int}$	$T.tipo := ENTERO$
$L \rightarrow \text{coma id } L$	$L_1.th := L.th; \text{anadirTS}(\text{id.lexema}, L.th)$
$L \rightarrow \epsilon$	

## Definiciones dirigidas por la sintaxis (DDS)

- Son un tipo de gramáticas de atributos.
- Las reglas incluyen acciones que no generan una traducción, sino que hacen definiciones (guardan información en la tabla de símbolos).
- También existen las **definiciones dirigidas por la sintaxis con atributos por la izquierda (DDSI)**, que son las que permiten la definición en una sola pasada.



## Definiciones dirigidas por la sintaxis (DDS)

- Las gramáticas de atributos y las DDS solamente **especifican las acciones** que es necesario realizar en cada regla, **pero no el orden** en el que hay que ejecutarlas.
- Son una herramienta para el diseño de alto nivel del traductor, sin entrar en los detalles de implementación.
- **El orden de ejecución de las acciones es muy importante.**

# ETDS

## Esquema de Traducción Dirigido por la Sintaxis – ETDS

Un esquema de traducción dirigido por la sintaxis (ETDS):

- es una gramática de atributos
- las acciones semánticas se insertan en la parte derecha de las reglas **en el momento exacto del análisis en el que se tienen que ejecutar**



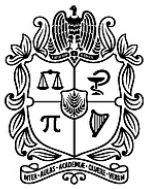
## Ejemplo 5 de traducción con ETDS

`int a,b,c`

NOMBRE	TIPO	DIRECCIÓN
a	ENTERO	0
b	ENTERO	2
c	ENTERO	4

### ESQUEMA DE TRADUCCIÓN DIRIGIDO POR LA SINTAXIS (ETDS)

$$\begin{aligned}
 D &\rightarrow T \text{ id } \{ \text{anadirTS}(\text{id.lexema}, T.tipo); L.th := T.tipo \} L \\
 T &\rightarrow \text{float } \{ T.tipo := REAL \} \\
 T &\rightarrow \text{int } \{ T.tipo := ENTERO \} \\
 L &\rightarrow \text{coma id } \{ \text{anadirTS}(\text{id.lexema}, L.th); L_1.th := L.th \} L \\
 L &\rightarrow \epsilon
 \end{aligned}$$




## Características de los ETDS

- Permiten diseñar **traductores de una sola pasada**
- Deben cumplir las siguientes **restricciones**:
  - ✓ Un atributo heredado de un símbolo  $i$  de la parte derecha de la regla se debe calcular en una acción semántica situada antes de  $i$   
Ejemplo:  $D \rightarrow T \text{ id } \{ \dots; L.th := T.tipo \} L$
  - ✓ Una acción semántica **no** puede referirse a un atributo sintetizado de un símbolo situado a la derecha de la acción en la regla  
Ejemplo:  $D \rightarrow \text{id } \{ \dots; L.th := T.tipo \} L T$
  - ✓ Un atributo sintetizado de la parte izquierda de una regla sólo se puede calcular después de haber calculado todos los atributos que se usan para calcularlo (preferiblemente al final de la regla)



## Ejemplo 3 de traducción (con ETDS)

`int a, b`  `a: integer; b: integer`

### GRAMÁTICA DE ATRIBUTOS

REGLA	ACCIÓN SEMÁNTICA
$D \rightarrow T \text{ id } L$	$L.th := T.trad; D.trad := \text{id.lexema}    " : "    T.trad    L.trad$
$T \rightarrow \text{float}$	$T.trad := \text{"real"}$
$T \rightarrow \text{int}$	$T.trad := \text{"integer"}$
$L \rightarrow \text{coma id } L$	$L_1.th := L.th; L.trad := ";"    \text{id.lexema}    " : "    L.th    L_1.trad$
$L \rightarrow \epsilon$	$L.trad := ""$

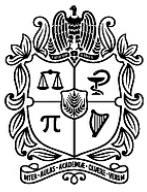
### ESQUEMA DE TRADUCCIÓN DIRIGIDO POR LA SINTAXIS

$D$	$\rightarrow$	$T \text{ id } \{ L.th := T.trad \} L \{ D.trad := \text{id.lexema}    " : "    T.trad    L.trad \}$
$T$	$\rightarrow$	$\text{float } \{ T.trad := \text{"real"} \}$
$T$	$\rightarrow$	$\text{int } \{ T.trad := \text{"integer"} \}$
$L$	$\rightarrow$	$\text{coma id } \{ L_1.th := L.th \} L \{ L.trad := ";"    \text{id.lexema}    " : "    L.th    L_1.trad \}$
$L$	$\rightarrow$	$\epsilon \{ L.trad := "" \}$

## Implementación de un ETDS

Mediante un **analizador descendente recursivo**:

- Los ***atributos sintetizados*** deben ser retornados por los métodos de los no terminales (cuando hay más de un atributo es mejor retornar un objeto con todos los atributos)
- Los ***atributos heredados*** son parámetros que son pasados como argumentos a los métodos de los no terminales
- **IMPORTANTE:** es necesario almacenar el lexema de los tokens antes de llamar a la función `emparejar`
- La traducción de la cadena de entrada es retornada por la función que analiza el símbolo inicial de la gramática



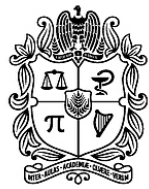
## Ejemplo 3 de traducción, implementado con un ASDR

$D \rightarrow T \text{ id } \{ L.th := T.trad \} L \{ D.trad := \text{id.lexema} \parallel ":" \parallel T.trad \parallel L.trad \}$

```
String D()      // D -> T id L
{
    String ttrad, idlexema, ltrad;

    ttrad = T();
    idlexema = lexema;
    empareja(ID);
    ltrad = L(ttrad);    // L.th := T.trad

    return idlexema + ":" + ttrad + ltrad;
}
```



## Ejemplo 3 de traducción, implementado con un ASDR

```
String L(String th)  // L -> coma id L  | epsilon
{
    if (token == COMA)
    {
        String idlexema, ltrad;

        emparejar(COMA);
        idlexema = lexema;
        emparejar(ID);
        ltrad = L(th);  // L1.th := L.th
        return ";" + idlexema + ":" + th + ltrad;
    }
    else if (token == FINFICHERO)
        return "";  // L -> epsilon { L.trad := "" }
    else
        errorSintactico(. . . .);
}
```





## Traducción de expresiones aritméticas con ETDS

$a+b-c+d$

$a+b-c$



$\text{sum}(\text{res}(\text{sum}(a, b), c), d)$

$\text{res}(\text{sum}(a, b), c)$

La asociatividad por la izquierda de los operadores '+' y '-' implica utilizar una gramática con recursividad por la izquierda:

$E \longrightarrow E \text{ op } T$

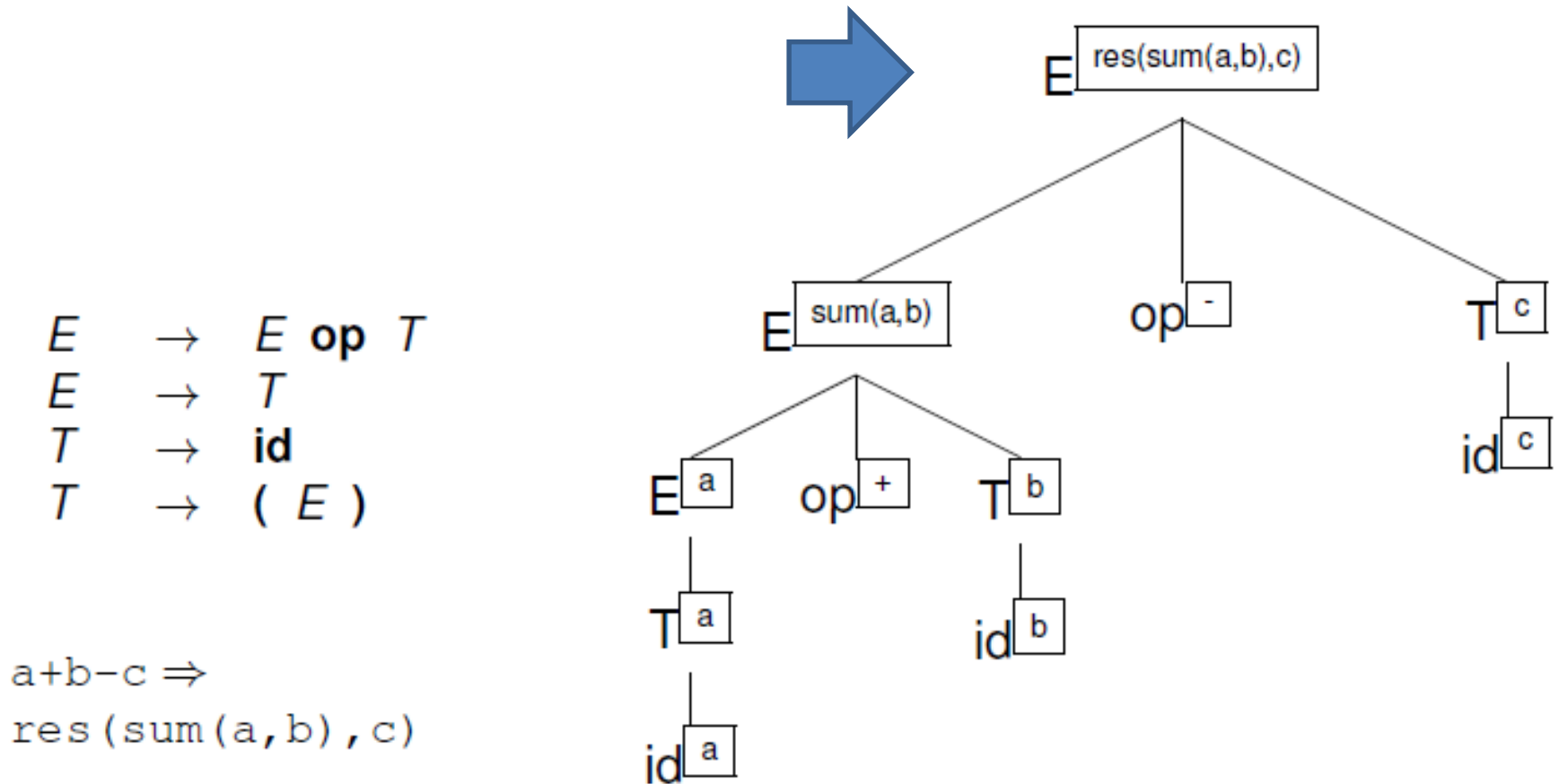
$E \longrightarrow T$

$T \longrightarrow \text{id}$

$T \longrightarrow ( E )$



## Traducción de expresiones aritméticas con ETDS





## Traducción de expresiones aritméticas con ETDS

El ETDS quedaría así:

$$\begin{aligned} E &\longrightarrow E \text{ op } T \{ E.trad := \text{op}.trad || "(" || E_1.trad || ", " || T.trad || ")" \} \\ E &\longrightarrow T \{ E.trad := T.trad \} \\ T &\longrightarrow \text{id} \{ T.trad := \text{id.lexema} \} \\ T &\longrightarrow ( E ) \{ T.trad := E.trad \} \end{aligned}$$



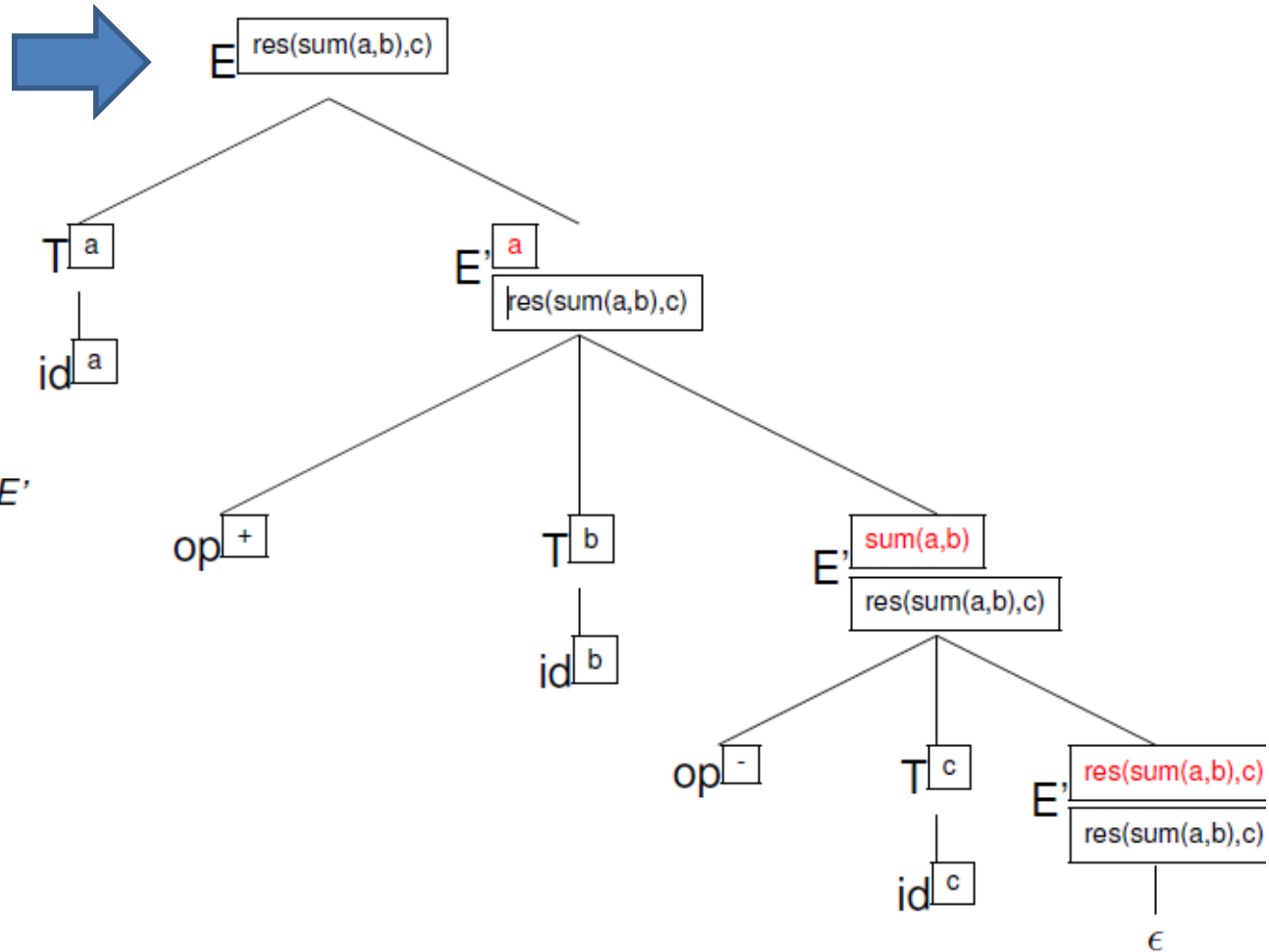
## Traducción de expresiones aritméticas con ETDS

Si queremos realizar la misma traducción con una gramática sin recursividad por la izquierda (si queremos usar un ASDR), la gramática sería:

$$\begin{aligned} E &\longrightarrow T E' \\ E' &\longrightarrow \text{op } T E' \\ E' &\longrightarrow \epsilon \\ T &\longrightarrow \text{id} \\ T &\longrightarrow ( E ) \end{aligned}$$



## Traducción de expresiones aritméticas con ETDS



$E \rightarrow T E'$   
 $E' \rightarrow \text{op } T E'$   
 $E' \rightarrow \epsilon$   
 $T \rightarrow \text{id}$   
 $T \rightarrow ( E )$

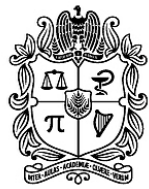
$a+b-c \Rightarrow$   
 $\text{res}(\text{sum}(a,b),c)$



## Traducción de expresiones aritméticas con ETDS

El ETDS quedaría así:

$$\begin{aligned} E &\longrightarrow T \{ E'.th := T.trad \} E' \{ E.trad := E'.trad \} \\ E' &\longrightarrow \mathbf{op} \ T \{ E'_1.th := \mathbf{op}.trad || "(" || E'.th || "," || T.trad || ")" \} \\ &\quad E' \{ E'.trad := E'_1.trad \} \\ E' &\longrightarrow \epsilon \{ E'.trad := E'.th \} \\ T &\longrightarrow \mathbf{id} \ { T.trad := \mathbf{id}.lexema \} \\ T &\longrightarrow ( \ E \ ) \ { T.trad := E.trad \} \end{aligned}$$



## Ejercicio 1 (1/2)

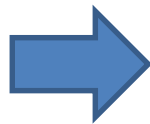
Diseñar un ETDS para traducir declaraciones de funciones en C a Pascal. El proceso de traducción se puede especificar con los siguientes ejemplos de traducción:

```
int f(void),  
    g(float a,int *b);
```

```
void h(int a,float *c),  
    j(void);
```

```
float f(int a);
```

```
int f(int a,int b,int c),  
    g(int d),  
    h(int e);
```

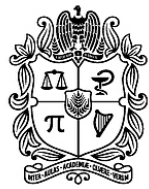


```
function f:integer;  
function g(a:real;var b:integer):integer;
```

```
procedure h(a:integer;var c:real);  
procedure j;
```

```
function f(a:integer):real;
```

```
function f(a:integer;b:integer;c:integer):integer;  
function g(d:integer):integer;  
function h(e:integer):integer;
```



## Ejercicio 1 (2/2)

Utilizar como base la siguiente gramática, que genera el lenguaje fuente:

<i>S</i>	→	<i>TipoFun</i> <i>L</i> <b>puntoycoma</b>
<i>TipoFun</i>	→	<b>void</b>
<i>TipoFun</i>	→	<b>int</b>
<i>TipoFun</i>	→	<b>float</b>
<i>L</i>	→	<i>F</i> <i>Lp</i>
<i>Lp</i>	→	<b>coma</b> <i>F</i> <i>Lp</i>
<i>Lp</i>	→	ε
<i>F</i>	→	<b>ident</b> <i>lpar</i> <i>A</i> <i>rpar</i>
<i>A</i>	→	<b>void</b>
<i>A</i>	→	<i>Argu</i> <i>M</i>
<i>M</i>	→	<b>coma</b> <i>Argu</i> <i>M</i>
<i>M</i>	→	ε
<i>Argu</i>	→	<i>Tipo</i> <b>ident</b>
<i>Argu</i>	→	<i>Tipo</i> <b>asterisco</b> <b>ident</b>
<i>Tipo</i>	→	<b>int</b>
<i>Tipo</i>	→	<b>float</b>

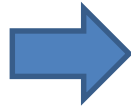




## Ejercicio 2 (1/2)

Diseñar un ETDS para traducir declaraciones de funciones y procedimientos anidados en Pascal a C. El proceso de traducción se puede especificar con los siguientes ejemplos:

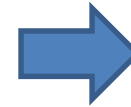
```
function f:integer;  
  codigo  
endfunc;
```



```
int f();  
  
void p1();  
void p1_p2();
```

```
procedure p1;  
  procedure p2;  
    codigo  
  endproc;  
  codigo  
endproc;
```

```
procedure p1;  
  procedure p2;  
    codigo  
  endproc;  
  function f1:real;  
    codigo  
  endfunc;  
  procedure p3;  
    procedure p4;  
      codigo  
    endproc;  
    codigo  
  endproc;  
  codigo  
endproc;
```



```
void p1();  
void p1_p2();  
float p1_f1();  
void p1_p3();  
void p1_p3_p4();
```



## Ejercicio 2 (2/2)

Diseñar el ETDS utilizando como base la siguiente gramática, que genera el lenguaje fuente:

$$\begin{aligned} S &\longrightarrow P \\ P &\longrightarrow \text{procedure id ; } L \text{ endproc ;} \\ P &\longrightarrow \text{function id : } T \text{ ; } L \text{ endfunc ;} \\ L &\longrightarrow P L \\ L &\longrightarrow \text{codigo} \\ T &\longrightarrow \text{integer} \\ T &\longrightarrow \text{real} \end{aligned}$$