

# PDL

## ▼ Tema 1

- Lenguajes Naturales
  - Español
- Lenguajes Formales
  - lenguaje cuyos símbolos primitivos y reglas para unir esos símbolos están formalmente especificados

Lenguajes de programación.

- Niveles de lenguajes
  - Maquina
  - Ensamblador
  - Alto nivel
  - Orientados a problemas

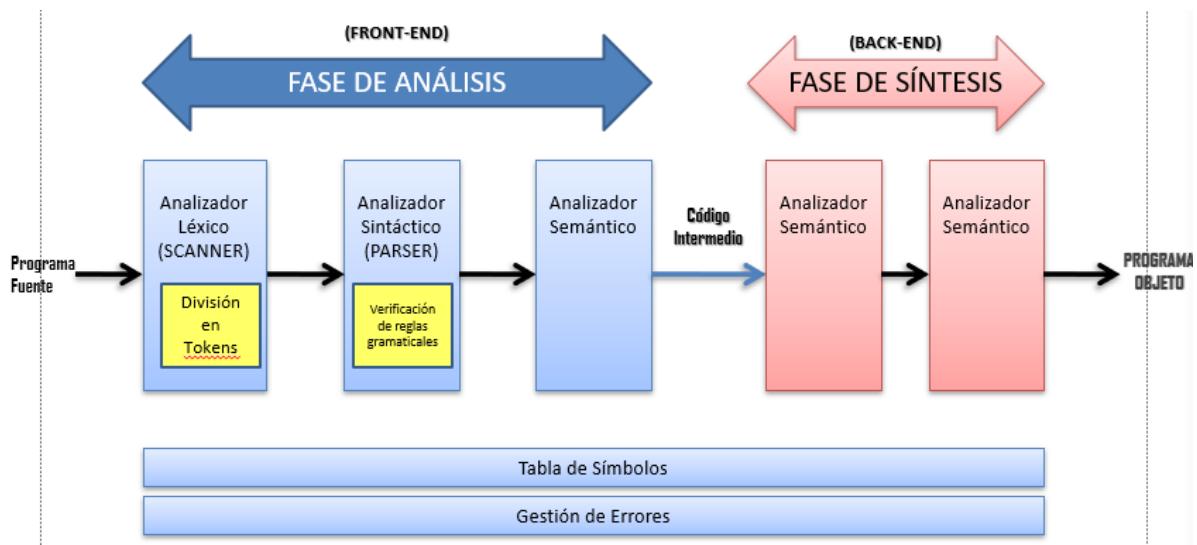
- Traductor

Programa que lee un programa en lenguaje fuente de alto nivel y lo traduce a lenguaje objeto

- Intérprete
 

Ejecuta directamente lo que traduce
- Compilador
 

Genera un fichero del lenguaje objeto que luego se ejecutará



- **Analizador Léxico**

Es un programa capaz de descomponer una entrada de caracteres en una secuencia ordenada de tokens.

- **Token**

Secuencia de caracteres con significado sintáctico propio.

- **Lexema**

Secuencia de caracteres cuya estructura se corresponde con el patrón de un token

- **Patrón**

Regla que describe los lexemas correspondientes a un token

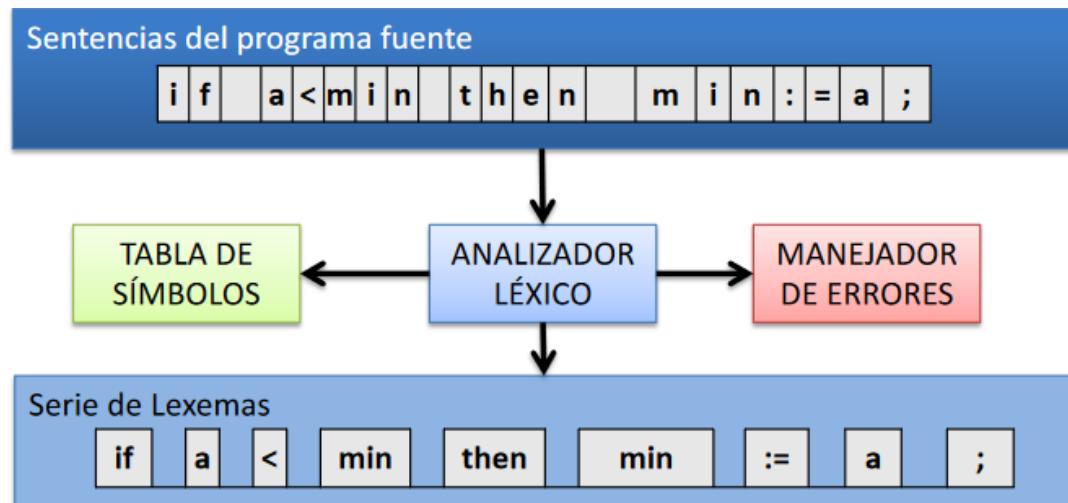
- Tareas

Reconocer los componentes léxicos en la entrada.

Eliminar espacio, tab,...

Eliminar comentarios de entrada

Detectar errores léxicos



**IF X < 10 THEN X := X + y**

<79,-> <12,32> <28,-> <13,10> <80,-> <12,32> <65,->  
<12,32> <34,-> <12,33>

Es decir:

<if,-> <identificador,32> <op\_menor,-> <literal\_entero,10> <then,->  
<identificador,32> <asignación,-> <identificador,32> <op\_suma,->  
<identificador,33>

- Alfabeto

$$\Sigma_1 = \{0, 1\}$$

$$\Sigma_2 = \{a, b\}$$

$$\Sigma_3 = \{na, pa, bra, la\}$$

$$\Sigma_4 = \{\langle HTML \rangle, \langle /HTML \rangle, \langle BODY \rangle, \langle /BODY \rangle, \dots\}$$

- **Expresiones regulares**

→ 0 ó más →  $a^*$

→ 1 ó más →  $aa^* \equiv a^+$

→ 0 ó 1 →  $\epsilon \mid a \equiv a?$

→ Clases de caracteres →  $a \mid b \mid c \equiv [abc]$ . También:  $a \mid b \mid c \mid \dots \mid z \equiv [a-z]$

→ Los paréntesis agrupan subexpresiones:  $(a \mid b \mid c)^*$

▼ Tema 2

- **Lenguaje**

Cualquier subconjunto del universo sobre algún alfabeto.

- Lenguaje vacío
- Lenguaje que solo tenga la palabra vacío
- $\{a, b, \epsilon\}$

- **Producción y derivaciones**

ejm

000 → 010 y 10 → 01 son 2 producciones

Dado 1000

1010 es una producción

0100 es una producción

Una **regla de producción** produce derivaciones

- **Lenguaje regular**

Describe una familia de tokens que se ajustan a un determinado patrón léxico.

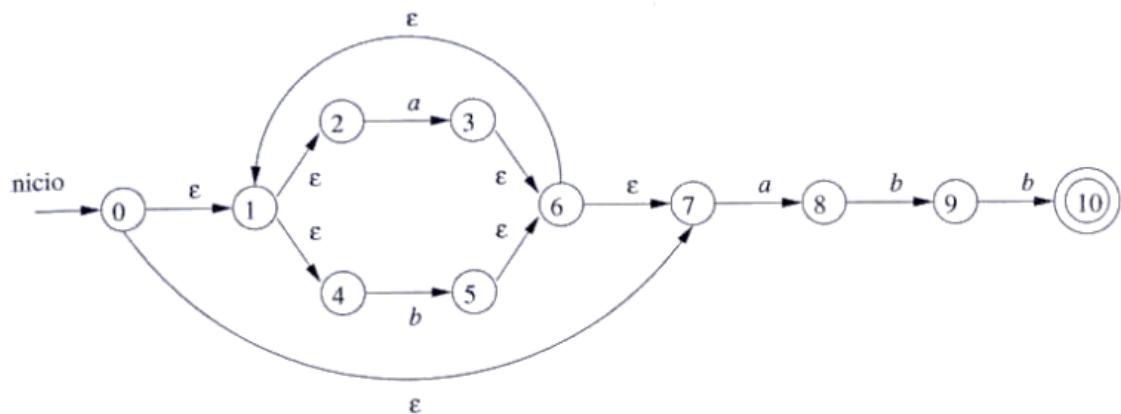
- Definiciones regulares

Expresiones regulares con nombre  
ejm

Dígito → [0-9]

- Autómata finito

reconoce una cadena de entrada si consumimos todos los caracteres de la misma cumple con la ER



ESTADO DEL AFN	ESTADO DEL AFD	<i>a</i>	<i>b</i>
{0, 1, 2, 4, 7}	<i>A</i>	<i>B</i>	<i>C</i>
{1, 2, 3, 4, 6, 7, 8}	<i>B</i>	<i>B</i>	<i>D</i>
{1, 2, 4, 5, 6, 7}	<i>C</i>	<i>B</i>	<i>C</i>
{1, 2, 4, 5, 6, 7, 9}	<i>D</i>	<i>B</i>	<i>E</i>
{1, 2, 3, 5, 6, 7, 10}	<i>E</i>	<i>B</i>	<i>C</i>

### ▼ Tema 3

En el analizador sintáctico cuando veamos una letra, NO es una letra, sino un token pero por facilidad está abreviado

Los nodos hoja → en minúscula

nodos de expresión → en Mayúscula

una gramática tiene los tokens

las reglas de la gramática (nodos intermedios)

en nodo raíz de todo el árbol

reglas para generar la gramática

## Componentes de una gramática: $G = (V_t, V_n, S, P)$

- Símbolos terminales(componentes léxicos [tokens]),  $V_t$
- Símbolos no-terminales,  $V_n$
- Símbolo inicial o axioma,  $S$
- Producciones, formadas por no-terminales y terminales,  $P$

- Tipos de analizador sintáctico

- Descendente

- ▼ Análisis recursivo

- ejm cadena ccd

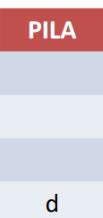
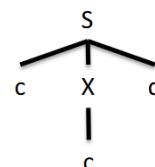
$$G(S \rightarrow c X d ; X \rightarrow c k | c)$$

Regla  
 $X \rightarrow c k$

Entrada

c	c	d
---	---	---

↑



ccd para  $G(S \rightarrow c X d ; X \rightarrow c k | c)$

Entrada	Pila	Regla a aplicar	Se obtiene en la pila	Se empareja con la entrada	Queda en la pila	Queda en la entrada
c c d	S	$S \rightarrow c X d$	c X d	c	X d	c d
c d	X d	$X \rightarrow c k$	c k d	c	k d	d
d	d k	d y k no se emparejan, hay que deshacer el último paso (2) y probar la siguiente regla				
c d	X d	$X \rightarrow c$	c d	c	d	d
d	d			d		

- ▼ Análisis predictivo

## LL(k) ejm LL(1)

k me dice cuantos token extra tengo que mirar

Creo una tabla con todas las combinaciones posibles y solo pruebo las estén en la tabla

2. Si  $\exists A \rightarrow \alpha B \beta \Rightarrow \{\text{Primero}(\beta) - \epsilon\} \in \text{Sigiente}(B)$
3. Si  $\exists A \rightarrow \alpha B$  ó  $(A \rightarrow \alpha B \beta \text{ donde } \epsilon \in \text{Prim}(\beta)) \Rightarrow \text{Sig}(B) \subseteq \text{Sig}(A)$

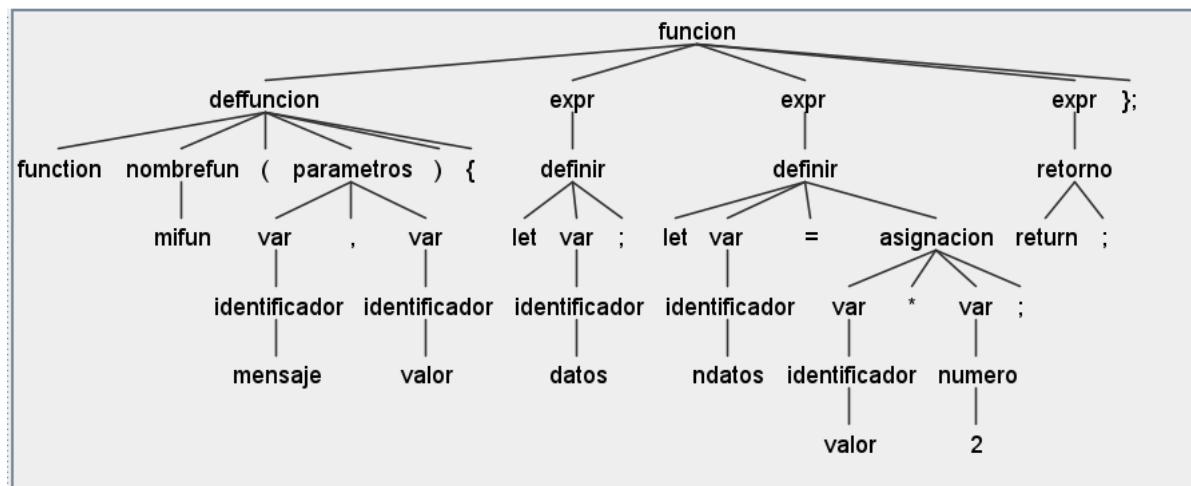
### REGLAS de construcción de la Tabla de Análisis (M [Vn,Vt] )

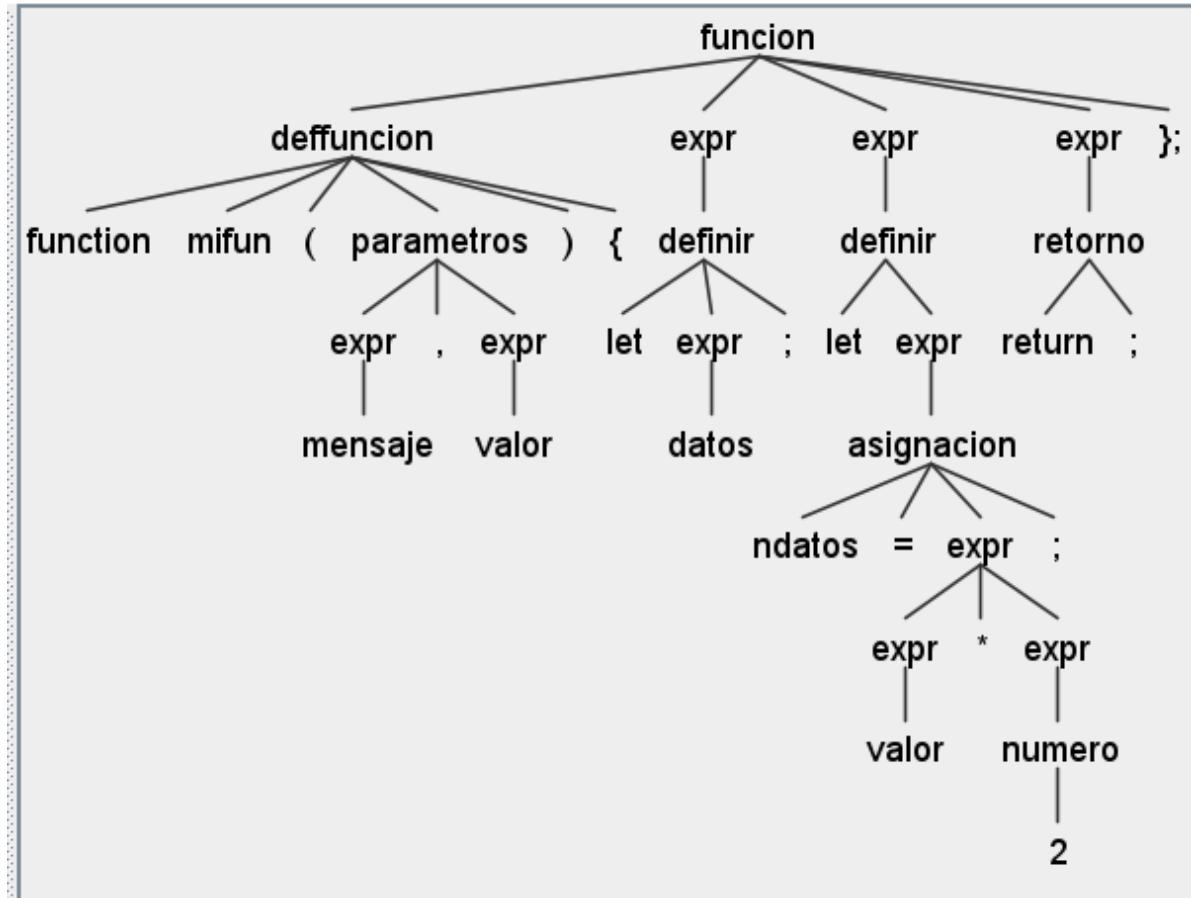
Repita todo este proceso para cada producción  $A \rightarrow \alpha$  de la gramática

1. Para cada terminal  $a \in \text{Primero}(\alpha)$ :
  - Agregue  $A \rightarrow \alpha$  a la tabla M[A,a]
2. Si  $\epsilon \in \text{Primero}(\alpha)$  (porque  $\alpha$  deriva  $\epsilon$ ) :
  - Agregar la producción  $A \rightarrow \alpha$  en la posición M[A,b] de la tabla para cada terminal  $b \in \text{Sigiente}(A)$ .
3. Si  $\epsilon \in \text{Primero}(\alpha)$  y  $\$ \in \text{Sigiente}(A)$  :
  - Agregue  $A \rightarrow \alpha$  en la posición M[A,\$]

- ascendente

### ▼ Ejemplos arboles laboratorio





## Apuntes teoría Examen 2



- La tabla de símbolos

**Es una estructura global utilizada por distintos módulos del compilador**

Contiene una entrada para cada uno de los símbolos definidos en el programa fuente

operaciones

Insertar, buscar y eliminar

3 estructuras de implementación

- listas lineales (simple o doble enlazadas)

Es un sistema sencillo pero lento ante muchas entradas

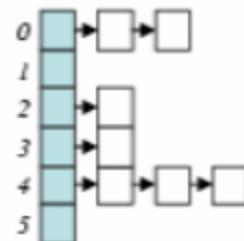
- tablas de dispersión

Es un array con entradas indexadas

Es la más utilizada

Una función de dispersión convierte el nombre del identificador en un valor que corresponde con un índice.

En caso de colisión, se realiza el encadenamiento por separación



- arboles de búsqueda binaria

2 tipos de declaraciones

- Explícitas
- Implícitas

Hay cuatro clases de declaraciones EXPLÍCITAS:

- De constante: **const double PI = 3.1415;**
- De tipo: **struct Complejo{ float re, im; }**
- De variable: **bool encontrado;**
- De función: **void f(int,float);**

Declaraciones IMPLÍCITAS: en Fortran un identificador no declarado que comience por una letra entre la *i* y la *n* es un entero.

- Reglas de ámbito

Varia según el lenguaje pero existen algunas comunes a todas

- Declarar antes de usar
- Estructura de bloques

Se permite la anidación de bloques dentro de otros

```
float g(int a, floatb){  
    int x, y;  
    for (x=0;x<10;x++){  
        bool encontrado;  
        //...  
        {  
            char* x;  
        }  
    }  
}
```

- Anidación más próxima
  - Alternativas de implementación
    - Construir una nueva tabla de símbolos para cada ámbito
    - Puede llegar a bastar con numerar los ámbitos por niveles
- 

- Semántica

Tenemos ya el conjunto de tokens, pero no sabemos nada acerca de su significado

Conjunto de reglas que especifican el significado de cualquier sentencia correcta y escrita en un determinado lenguaje

Tareas que componen el proceso del análisis semántico

- Fase de obtención de info necesaria para la compilación tras conocer la estructura sintáctica
- Completar las fases de análisis léxico y sintáctico incorporando comprobaciones
- Identificar cada tipo de instrucción y sus componentes
- Completar la tabla de símbolos
- Realizar comprobaciones estáticas
- Realizar comprobaciones dinámicas
- validar las declaraciones de identificadores

El análisis se divide en 2 categorías

- Análisis de la exactitud del programa

Para garantizar una ejecución adecuada

- Análisis para mejorar la eficiencia

Optimización

- No existe una notación estandar para especificar la semántica estática

Estas pueden ser formal o informal

- Gramática de atributos

es una GLC cuyos símbolos pueden tener asociados atributos

Autómata a pila + acciones semánticas = Traductor dirigido por la sintaxis

Tipo	Ejemplo	
Declaración de constantes	<code>CONST MAX = 100;</code>	Registrar 'MAX' como constante entera de valor 100
Declaración de tipos	<code>TYPE TVector = ARRAY [1..MAX] OF INTEGER;</code>	Registrar 'Tvector' como tipo ARRAY de tamaño 100 y base INTEGER
Declaración de variables	<code>VAR v : TVector;</code>	Comprobar que 'Tvector' existe como un tipo y registrar 'v' como una variable de ese tipo
Declaración de procedimientos	<code>PROCEDURE Sort (VAR v : TVector);</code>	Registrar 'Sort' como un procedimiento del ámbito en curso indicando la lista de tipos de los parámetros. Crear un nuevo ámbito y registrar en él 'v' como variable de tipo 'Tvector'

Existen 2 Notaciones para asociar reglas semánticas

- Definiciones Dirigidas por sintaxis (DDS)
  - Esquemas de traducción (EDT)
- Atributo

Propiedad de una construcción de un lenguaje

Cada símbolo (terminal o no) puede tener asociado un número finito de atributos

- Fijación de un atributo

Proceso de calcular el valor de un atributo y asociarlo con una construcción del lenguaje

puede ser estático o dinámico

Se denotan de la siguiente forma

**NombreSímbolo.NombreAtributo**

Ejemplo:

```
numero → numero digito | digito
a) numero → digito
   numero.valor = digito.valor
b) numero → numero digito
   numero1.valor = numero2.valor * 10 + digito.valor
```

Otra notación hace referencia a su posición en la regla de producción.

Ejemplo:

```
numero → numero digito | digito
a) numero → digito
   $$ . valor = $1 . valor
b) numero → numero digito
   $$ . valor = $1 . valor * 10 + $2 . valor
```

- Gramática de atributos

Se escriben en forma de tabla

Regla Gramatical	Regla Semántica
$L \rightarrow E\ n$	<code>print(E.val)</code>
$E \rightarrow E + T$	<code>E<sub>0</sub>.val = E<sub>1</sub>.val + T.val</code>
$E \rightarrow T$	<code>E.val = T.val</code>
$T \rightarrow T * F$	<code>T<sub>0</sub>.val = T<sub>1</sub>.val * F.val</code>
$T \rightarrow F$	<code>T.val = F.val</code>
$F \rightarrow (E)$	<code>F.val = E.val</code>
$F \rightarrow \text{digito}$	<code>F.val = digito.valor_lexico</code>

- DDS

Tipos de atributos

- Sintetizados

su valor se calcula en función de atributos de nodos hijos

- Heredados

Para un hijo se calculan a través de los atributos del padre

Las reglas semánticas establecen las dependencias entre los atributos

Se representan con un grafo de dependencia o con un árbol sintáctico con anotaciones

## Forma de una Definición Dirigida por la Sintaxis:

- I. Cada producción  $A \rightarrow \alpha$  tiene una o más reglas semánticas asociadas.
- II. Cada regla tiene la forma  $b = f(c_1, c_2, \dots, c_n)$ .
- III.  $b$ , que depende de  $c_1, c_2, \dots, c_n$ , puede ser:
  - a) Un atributo sintetizado de  $A$ .
  - b) Un atributo heredado de uno de los símbolos del lado derecho de la producción.
- IV. Las funciones  $f$  de las reglas se escriben como expresiones.

- Gramática S-atribuidas

todos los atributos asociados con los símbolos son sintetizados

Evaluación de atributos sintetizados

Los valores de los atributos S se pueden calcular fácilmente mediante un recorrido ascendente (**post-orden**) del árbol sintáctico:

```
procedimiento EvaluarSintetizado(A:arbolSintactico){  
    Para cada hijo H de A hacer  
        EvaluarSintetizado(H);  
    Calcular atributos sintetizados de A  
}
```

Ejemplo

$L \rightarrow E\ n$	{ print ( $E_1.val$ ) } (* $n$ = salto línea *)
$E \rightarrow E + T$	{ $E_0.val = E_1.val + T_3.val$ }
$E \rightarrow T$	{ $E_0.val = T_1.val$ }
$T \rightarrow T * F$	{ $T_0.val = T_1.val * F_3.val$ }
$T \rightarrow F$	{ $T_0.val = F_1.val$ }
$F \rightarrow (E)$	{ $F_0.val = E_2.val$ }
$F \rightarrow \text{digito}$	{ $F_0.val = \text{digito}$ }

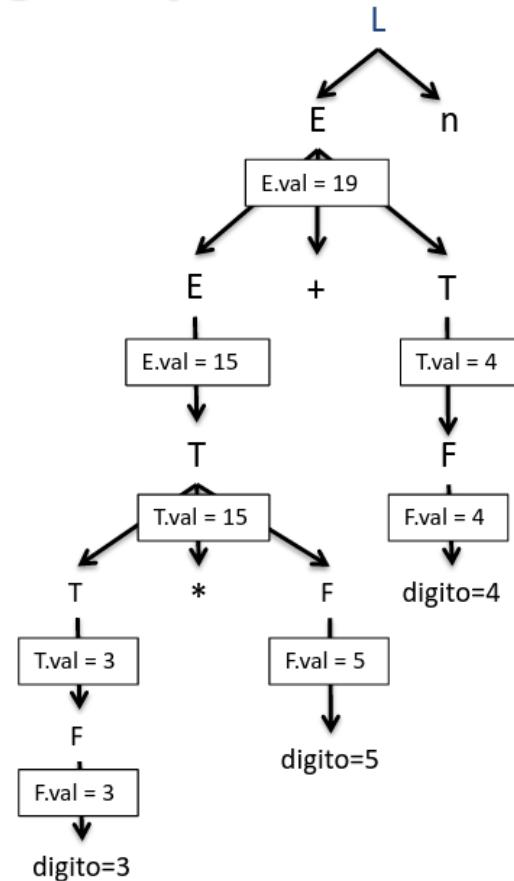
## DEFINICIONES DIFERENCIAS ENTRE LA GRAMÁTICA L-ATRIBUIDAS Y LA GRAMÁTICA S-ATRIBUIDAS

### Gramática S-atribuidas

Evaluación de la expresión :  
“3 \*5 + 4”

El resultado se imprime

$L \rightarrow E\ n$
$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow T * F$
$T \rightarrow F$
$F \rightarrow (E)$
$F \rightarrow \text{digito}$



- Gramática L-atribuidas

Los atributos asociados son Sintetizados o Heredados

Los valores de los atributos S se pueden calcular mediante un recorrido ascendente (pre-orden) del árbol sintáctico

### Gramática L-atribuida

Ejemplo de gramática L-Atribuida:

```

D → T L { L.her = T.tipo; }
T → int { T.tipo = entero; }
T → real { T.tipo = real; }
L → L , id { L1.her = L0.her; añadetipo(id,L0.her); }
L → id { añadetipo(id, L.her); }

```

Evaluar la expresión:

“real id1, id2, id3”

