

# Apuntes-PDL-lexico-sintactico-se...



Anónimo



Procesadores de Lenguajes



3º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenieros Informáticos  
Universidad Politécnica de Madrid



**Descarga la APP de Wuolah.**  
Ya disponible para el móvil y la tablet.





# Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the  
App Store

GET IT ON  
Google Play



18

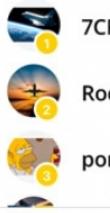
Ver mis op

Continúa de



405416\_arts\_escuela2016juniy.pdf

Top de tu g



7CR



Rocio



pony



Inicio



Asign

## 1. Introducción - Compiladores

Compilación: proceso de traducción que convierte un programa escrito en un lenguaje (Sav, C...) a un programa equivalente en otro lenguaje (ensamblador).

Intérprete: compilador paso a paso

Ensamblador: compilador de lenguaje ensamblador

Nº de pasos: nº de lecturas desde el inicio del fichero hasta el final. (Suf con uno)

## Lenguajes y gramáticas

④ Analizador sintáctico → gramática de tipo 2

Gramática (N, T, P, S)

N: símbolos no terminales → A, D.

T: símbolos terminales → a, for, \$.

P: reglas de producción → A → o D

S: axioma o símbolo inicial

⑤ Analizador léxico → gramática de tipo 3

## Fases del proceso de compilación

① Análisis léxico: lee el fichero, lo recorre carácter a carácter y extrae sus "palabras".

Ej: límite := largo \* alto - 1  
 límite <var, 1> → identificador = variable  
 \* <asig, >  
 largo <var, 2>  
 \* <POR, 1> como todos los se son iguales, no hay que escribir un nº para distinguirlos  
 alto <var, 3>  
 - <multos, >  
 \* <NUM, 1>, valor del nº en nuestro caso es 1

## Tabla símbolos

Fichero  
fuente

An. léxico  
uso políptico

An. semántico  
políptico

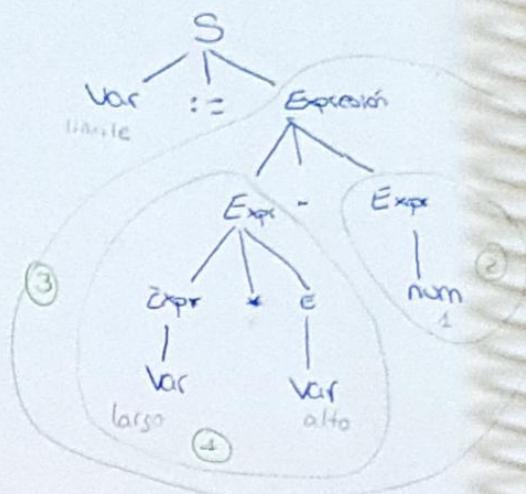
An. sintético  
árbol sintáctico

árbol  
sintáctico  
anotado

WUOLAH

② Análisis sintáctico: leer los tokens y agruparlos → generar el árbol sintáctico

- 1º Identifica <var, 2> } 3º
- 2º Identifica <var, 3> } 5º
- 4º Identifica <num, 55> } 6º
- <var, 55>
- <asig, >



③ Análisis semántico: recorre el árbol de arriba a abajo y de izq a derecha y comprueba si es correcto. Comprueba q. largo es una var de tipo int y que alto tb, y entonces ⑤ es correcto. Tb con ② - ④

### Fase de análisis

1. Generador de código intermedio: recibe el árbol sintáctico por a. semántico, lo traduce a un lenguaje intermedio más sencillo.

$$t_3 := \text{largo} \times \text{alto}$$

$$t_2 := \text{ent} - \text{real}$$

$$t_3 := t_2 - t_1$$

$$\text{límite} := t_3$$

2. Optimizador de código: optimiza código intermedio. → Código de más calidad.

$$t_2 := \text{ent} - \text{real} \quad | \quad t_3 = t_2 - \text{límite} \quad | \quad t_3 := \text{largo} \times \text{alto}$$

$$\text{límite} := t_3 - \text{límite}$$

3. Generador de código: genera las instrucciones y las traduce a ensamblado

MOV	largo, R1
MUL	alto, R1
MOV	R1, R2
MOV	Cs, R1
SUB	3.0, R1
MOV	R1, límite

4. Optimizador de código: optimiza el código final

5. Tabla de símbolos

Guarda la info

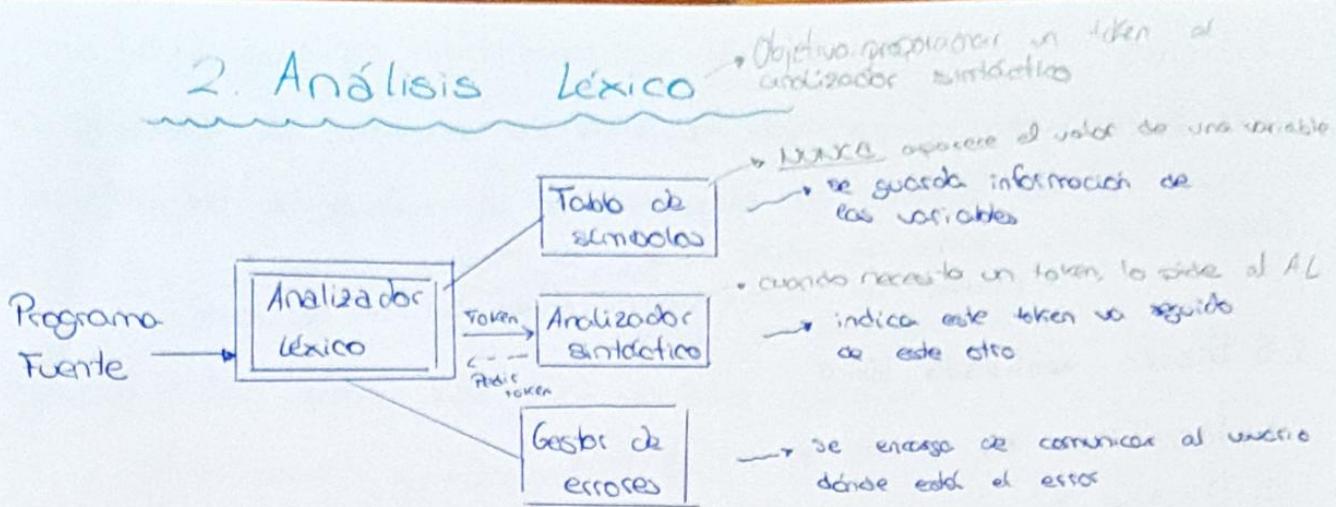
	Nombre	Tipo	dir. Mem
1	límite	real	30
2	largo	real	14
3	alto	real	28

dir. Mem que usará el código ensamblador cuando se ejecute



**KEEP  
CALM  
AND  
ESTUDIA  
UN POQUITO**

## 2. Análisis Léxico



### 2.1 FUNCIÓN DEL ANALIZADOR LÉXICO

- Encargado de leer el fichero fuente.
- Lee uno a uno los caracteres, y envía las palabras al analizador sintáctico en forma de **Token**. Token un token cuando leo carácter a carácter y llego a un espacio en blanco
- Se encarga de eliminar los elementos que no aportan información (espacios en blanco, salto de linea, comentarios...)
- Rellenar parte de la información en la **TS** → tabla de símbolos
- Relacionar errores con su posición en el texto fuente.

Línea de caracteres:  
empieza por comillas ""

Identificador: no empieza por comillas

$y=0$  → 3 tokens.  
→ 1 símbolo en la TS,  
6 x, que es el único identificador

- **TOKEN:** cada elemento del lenguaje que tiene significado propio (símbolos terminales)
- **Comentario NO es TOKEN**
- **Patrón:** regla o norma que nos dice cómo se escriben todas las palabras correspondientes a un determinado token.
- **Lexema:** secuencia de caracteres en el fichero de entrada que concuerda con un patrón de un determinado token.

④ **TOKEN** → **Tipo-Token:** análisis sintáctico  
Atributo: análisis semántico, traductor

<Tipo-Token, Atributo>

- Tipo de palabra
- Info adicional (cuando hay más de un lexema que concuerda con el patrón)
- Hecho que el análisis léxico

## 2.2 Errores

Error léxico cuando no se logra equiparar una secuencia de caracteres con alguno de los patrones del lenguaje  $\Rightarrow$  No se corresponde con un tipo

- Errores típicos:
- carácter no válido en el lenguaje  $\Rightarrow$  TOKEN válido
  - Nombre de usuario (identificadores, variables) no válido
  - Constante real no válida.

## 2.3 Diseño analizador léxico

1. Identificar los tokens de un lenguaje
2. Diseñar la gramática (Regular, Tipo 3, gramática que reconozca cada token)
3. Construir autómata finito determinista
4. Completar autómata con acciones semánticas
5. Identificar situaciones de error.

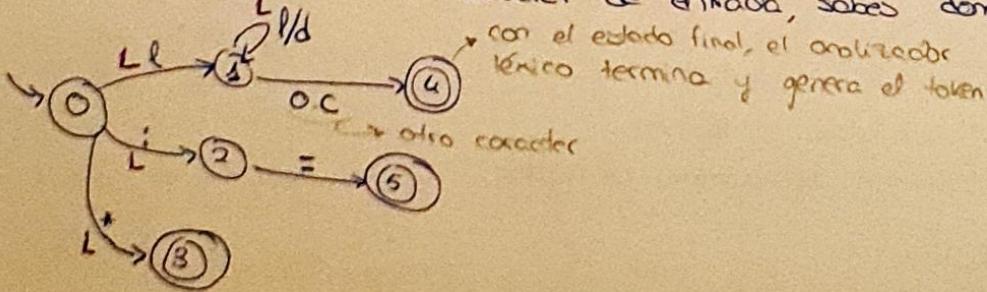
① Tokens Enteros y reales son dos tokens diferentes  
siempre se pone al "valor" al nº real o entero

② Gramática tipo 3  $\left\{ \begin{array}{l} A \rightarrow aB \xrightarrow{\text{terminal}} \\ A \rightarrow a \xrightarrow{\text{no terminal}} \end{array} \right.$   
No puede ser := no tiene 2 terminales (G2)  
 $A \rightarrow lBl : C | * | \text{del A}$  delimitador No genera token Salte  $\left\{ \begin{array}{l} \text{espacio en blanco} \\ \text{tabulador} \\ \text{fin de linea} \end{array} \right.$   
 $B \rightarrow lBl dB | \lambda$   
 $C \rightarrow =$

La Gramática Termina cuando se Genera un Token

## ③ AFD

\* Determinista: sabiendo el carácter de entrada, sabes dónde transita.



afdfa = auxiliar  
① → ④



# Descarga la APP de Wuolah.

## Ya disponible para el móvil y la tablet.

Available on the  
App Store

GET IT ON  
Google Play



18

Ver mis op.

Continúa d...



405416\_arts\_escuela2016juniy.pdf

Ejemplo: ¿ Cuántos tokens identificados por An. Léxico?

1) int indice = 1;  
1 2 3 4 5 → 5 tokens

2) while (n <= 30) { } → 8 tokens

3) c = b;  
1 2 3 4 → 4 tokens

④ cada uno de los elementos con significado propio.

Ejemplo: ¿ Son para TipoToken- atributo correctos?

Top de tu g...



7CR



Rocío



pony



Inicio

Asign.

1) < identificador, puntero >	si ✓
2) < constante - entera, >	NO X
3) < abre - parentesis, valor >	NO X
4) < punto - coma, >	si ✓
5) < identificador, >	NO X
6) < constante - entera, valor >	si ✓
7) < constante - real, valor >	si ✓
8) < while, >	si ✓
9) < while, valor >	NO X
10) < operador - aritmético, 2 >	si ✓
11) < lexema, puntero >	NO X
12) < operador - lógico, 3 >	si ✓
13) < operador - relacional, lexemas >	NO X
14) < operador - lógico, valor >	NO X → en la TS nunca se guarda el valor
15) < operador - relacional, 2 >	NO X
16) < punto - coma, punto - coma >	NO X
17) < FOR, >	SI ✓
18) < while, lexema >	NO X
19) < operador - relacional, 2 >	Si ✓

④ El analizador léxico siempre tiene que tener 2 carácter leído. (Sólo tiene memoria para 2 caracteres).

④ El analizador sintáctico combina el valor de zero. Declaración (que es una variable propia del an. sintáctico).

→ El carácter por el que transita se ha leído ya previamente en la acción anterior.

④ Nunca se puede encontrar una transición con λ si ese carácter no existe (no puede ser leído).

## Tabla de símbolos

Estructura de datos donde se almacena la información relevante sobre los identificadores del programa.

Todos los módulos del compilador tienen acceso.

- An. Léxico → función llamada del An. sintáctico que retorna un TOKEN
  - An. Léxico → tiene acceso al programa fuente. Le pasa un Token d. an. sintáctico
  - An. Sintáctico → comprobar si la estructura recibida del léxico es correcta.
- Cuando el A. Léxico lee un identificador, lo lleva a la (T.S) → una entrada para cada identificador
- An. Semántico → crea la TS y sella la información

Para cada entrada hay atributos

• Lexema: nombre del identificador. EL AN. LÉXICO SOLO METER EL LEXEMA.

• Tipo: entero, real, cadena, array... Dependiendo del tipo, se tienen unos atributos determinados.

UNA R linea que el parámetro se pasa por referencia y no por valor.

④ An. Léxico no sabe en qué tabla meter el lexema. No tiene visión de contexto. Eso lo hace el Analizador semántico

Operaciones que podemos hacer:

- 1) Insertar ID
- 2) Consultar ID
- 3) Crear TS vacía
- 4) Cambiar el valor de un atributo
- 5) Meter un atributo
- 6) Borrar la tabla

An. Léxico: añadir un nuevo lexema.

buscar si un lexema está en la tabla → comprobar si ya se creó una entrada para ese identificador.

Ejemplo: Real A, Aux, I → "he encontrado el identificador A y lo he puesto en la tabla T.S"  
Integer cont, x

Lexema	Tipo	...	Lexema	Tipo	...
A			A		
Aux			Aux		
I			cont		
cont			I		
X			X		

Analizador léxico: inserte los lexemas de los identificadores  
↓ Token

Analizador sintáctico  
↓ Árbol sintáctico

Analizador semántico

La TS tiene una entrada para cada identificada, y para cada entrada, una serie de atributos. En función del tipo de entrada (variable, array, función) habrá unos atributos determinados.

En general:

- Entrada, con los siguientes atributos:
  - Lexema (nombre identificado)
  - Tipo (entero, real, cadena)
  - Atributos en función del tipo

- ④ An. Léxico introduce el lexema en la TS.
- ⑤ An. Semántico crea/elimina TS (rellena los campos de atrib.).

## Gestión de la TS

- Crear una nueva TS (referenciarnos con un puntero).
- Gestión del ámbito o alcance } } Destruir una TS (liberar espacio que ocupa).

- Añadir información } } • Crear nueva entrada (añadir nuevo lexema) An. Léxico
- } } • Completar información de los atributos de una entrada existente

- Consultar información } } • Buscar un lexema: comprobación si se creó una entrada para un identificador An. Léxico
- } } • Consultar atributos de una entrada.

④ **Tuición Host:** recibe un parámetro, y el valor de salida (dirección propia) sea diferente para cada parámetro (entrada).

⑤ En lo práctico, podemos implementar la TS de una forma más sencilla.

### 3 TABLA DE SÍMBOLOS

El analizador léxico agrupa secuencias de caracteres (unidades mínimas de información), formando el lexema y codificar el token para pasárselo al Analizador Sintáctico.

El Analizador Léxico es una función llamada desde el Analizador Sintáctico que retorna un token.

El Analizador Semántico crea las Tablas de Símbolos y rellena la información del tipo de token insertado por el Analizador Léxico.

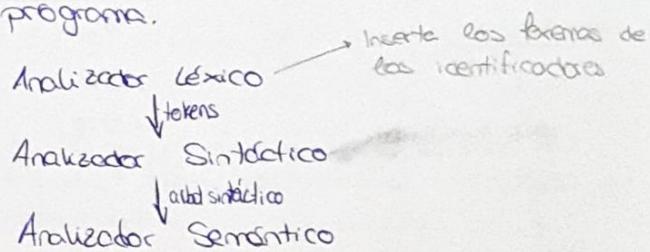
El campo Etiqueta de la TS es porque una función en ensamblador tendrá que tener una etiqueta para saltar a ella.

\* VAR indica que el parámetro se pasa por referencia y no por valor

1. Introducción Todos los módulos tienen acceso a ella *(consultar)* Almacena

La TS es una estructura de datos donde se almacena todo la información relevante sobre los identificadores del programa.

Lexema	Tipo	...



La TS tiene una entrada para cada identificador, y para cada entrada, una serie de atributos. En función del tipo de entrada (variables, arrays, función) tendrá unos atributos u otros.

En general, una entrada para cada identificador con unos campos para atributos

→ Lexema (nombre del identificador)

\* las entradas no son homogéneas  
pero todas tienen Lexema y tipo

→ Tipo (entero, real, cadena ...)

→ Atributos en función del tipo

\* An. Léxico introduce el Lexema en la Tabla Activa.

\* An. Semántico se encarga de crear, eliminar tablas.

\* Vamos a tener una TS para cada ámbito.



# Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the  
App Store

GET IT ON  
Google Play



122

18

Ver mis op

Continúa d

405416\_arts\_esce  
ues2016juniy.pdf

Top de tu g



7CR



Rocío



pony



Inicio

Asign.

## Gestión de la TS

- Gestión del ámbito o alcance } Crear una nueva TS → se refiere con un puntero
- Anadir información } Crear una nueva entrada ← ANADIR NUEVO LEXEMA An. Léxico
- } Completar información de los atributos de una entrada existente
- Consultar información } BUSCAR UN LEXEMA ← comprobar si ya se creó una entrada para ese identificador
- } Consultar atributos de una entrada.

\* Funció Hash: recibe un parámetro, y el valor de salida (dirección propia) sea diferente para cada parámetro (entrada)

### Ejemplo 1. TS

Real A, Aux, I

Integer cont, x

el des id encontrado está en la 3<sup>a</sup> pos, el 2<sup>a</sup> en la 2<sup>a</sup> pos...

#### 1) TS Lineal

Lexema Tipo ...

A

Aux

I

Cont

x

#### 2) TS ordenado (alfabéticamente)

Lexema Tipo ...

A

Aux

Cont

I

x

#### 3) TS hash con encadenamiento

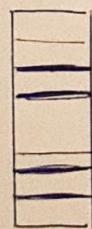
$$\text{Hash}(A)=27$$

$$\text{Hash}(\text{cont})=27$$

$$\text{Hash}(x)=27$$

$$\text{Hash}(\text{Aux})=42$$

$$\text{Hash}(I)=42$$



Lexema Tipo ... ... sig

A

Aux

I

Cont

x

enlazar AUX, I  
xq ambos tienen el  
mismo valor para  
la función HASH

Funció que da un valor diferente para cada id que es la pos de la TS ] HASH

El lexico aux Aplica HASH y le da 42

## Ejemplo: Tabla de simbolos

En las TS  $\xrightarrow{\text{An. Léxico introduce el lexema}}$

$\xrightarrow{\text{An. Semántico rellena los campos de atributos}}$

En este ejemplo, se supone:

- Lenguaje exige declaración previa de variables

- Tamaño de los tipos: entero 2B, dirección 8B y lógico 1B

- ★ Para gestionar las TS creadas y las activas, se suele crear una pila de punteros a las TS, con sus respectivos desplazamientos (para saber donde meter la siguiente entrada).
- ★ Sintáctico y Semántico trabajan simultáneamente.

### Ejecución

- 1) Léxico lee lexema, envía token al sintáctico
- 2) Sintáctico y Semántico trabajan simultáneamente
- 3) Sintáctico pide otro token al léxico
- 4) Léxico lee, oraño lexema (si no estaba ya metido)

→ El Token identificador puede ser:  $xid$ ,  $punteroTS$

→ PROCEDURE indica que entra en un nuevo ámbito (se activa el semántico para crear la tabla)

→ En proc, en la línea:

$x := b * 2 + c$  → no lo encuentra en la  $TS_{actual}$ , pero se va a la tabla del proceso padre y lo encuentra. (Para ello mira en la Pila de punteros o TS y desplazamiento).

→ Cuando se libera la TS, también se quita de la pila.

→ En el proc, cuando lee:  $fun(b)$ ; // da un error si no está en el ámbito de la TS activa.  
error: identificador no declarado

→ Para implementar errores

Fácil: que termine la ejecución

Difícil: que continúe hasta el final, y le posemos todos los errores

→ Desplazamiento: dirección relativa de las variables

↳ Para los nombres de función no hay desplazamiento

LAS ENTRADAS DE  
FUNCIONES NO  
TIENEN DESPLAZAMIENTO

↳ Ejecutar todo local  
y moverlo a todo del ámbito padre

Anidamiento de funciones: declarar una función hija dentro de una padre. proc

Puede usar variables locales (las suyas)

Puede usar variables globales (las de dentro de proc).

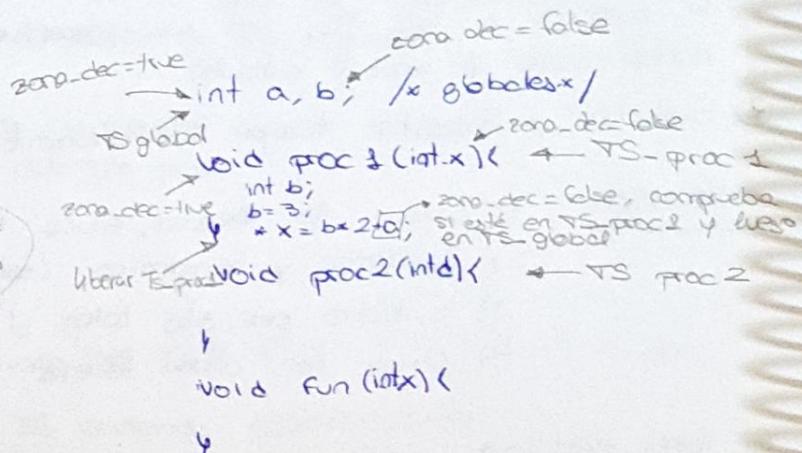
Puede usar variables del padre de su padre.

proc 2<  
  int a;  
  proc 3<  
    int b;  
    func 4<  
      int c;

## Ejemplo. TS sin anidamiento

- An. Semántico rellena los atributos Tipo y Despl de 'a' y 'b'.
- An. Semántico es el que cambia el flag zona-declaración.
- Cuando an. Léxico le pase el token <llave-izq>, > El semántico sabe que tiene que liberar la tabla de la función
- Los valores de los id (variables) no se guardan en la TS
- Buena práctica crear etiquetas que no se puedan confundir con otro token.

- ④ En la práctica no hay anidamiento (global y local) → Si hay while, habrá que crear una tabla para ese grupo de variables dentro del while  
↳ NO ocurre en la práctica



### TS Global

a	entero	0
b	entero	2
proc1	función	
proc2	función	

- ④ Ver cuando haya un id si está declarado en la TS-local (fun, proc1, proc2). Si no está, mirar en la TS global.

TS-proc1  
x Punteroentero  
b entero 0] → direcciones ocupan 2  
diferentes pd en memoria se manejan así cada

# Tema 4: Análisis Sintáctico

El Analizador Léxico es esclavo del Analizador Sintáctico

El Analizador Sintáctico comprueba que los tokens tienen una estructura sintáctica correcta.

Artículo sintáctico ② Tipo 2 } Usando gramática de contexto libre

Sin retroceso: saber en cada instante qué regla hay que aplicar (descendente)  
Con retroceso: varias reglas candidatas a expandir un nodo del árbol. Se elige una. Si no se consigue se retrocede hasta ese nodo y se aplica otra regla.

Dos tipos de Analizador Sintáctico

- Descendentes: del nodo raíz a los nodos hoja (Funciona por derivaciones a la izq)
- Ascendente: de las hojas a la raíz.

Base: secuencia de reglas que se han disparado y en qué orden.

Ejemplo: Analizador Sintáctico Descendente (Sin retroceso)

→ Funciona por derivaciones a la izquierda (coger el símbolo no terminal más a la izquierda)

1, 2, 4, 5, 3  
1 A → xpq  
2 x → pBq  
3 B → q  
4 B → CP  
5 C → m

→ Forma sentencial: cadena formada por símbolo no terminal y terminal (fijate los hojas del árbol)

Quando la forma sentencial está formada por símbolos terminales, ya he acabado el árbol.

④ La GCL (Gramática Contexto Libre) no puede ser recursiva por la izquierda porque nos metemos en un bucle infinito.

¿Cómo eliminamos la recursividad por la izq?

$$\begin{array}{l} A \rightarrow Ad \\ A \rightarrow B \end{array} \quad \left\{ \begin{array}{l} A \rightarrow BA' \\ A' \rightarrow \alpha A' \\ A' \rightarrow \lambda \end{array} \right\} \quad \text{Transformación}$$

Problema  
 $E \rightarrow E + T$        $\alpha$   
 $E \rightarrow T$        $\beta$

Problema  
 $T \rightarrow T * F$        $\alpha$   
 $T \rightarrow F$   
 $F \rightarrow id$

$E \rightarrow \overbrace{T}^{\alpha} E'$   
 $E' \rightarrow \overbrace{+ T}^{\beta} E' | \lambda$   
 $T \rightarrow \overbrace{F}^{\alpha} T'$   
 $T' \rightarrow * F T' | \lambda$   
 $F \rightarrow id$

T = Símbolos terminados minúsculas

N = No terminados mayúsculas



# Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the  
App Store

GET IT ON  
Google Play



122

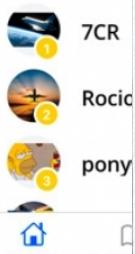
18

Ver mis op

Continúa d

405416\_arts\_esce  
ues2016juniy.pdf

Top de tu g



Top de tu g

## • Descendentes sin retraso

El siguiente token determina cuál es la regla a aplicar  $\rightarrow$  Geométricas LL(1)

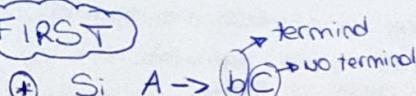
- ~ no terminal o expandir? Nos lo dice la técnica de derivación a la izq
- ~ ¿Regla a aplicar? Nos lo dice el siguiente token (gramáticas LL(1))

\*w es el axioma, pero si no dicen, el axioma es el primer no terminal  
 $\xrightarrow{\text{sig-term}}$  en la lista de reglas

FIRST: primeros símbolos terminales que podemos conseguir. (Algoritmo P.5)

Follow: símbolo terminal puede estar detrás/continuación de un símbolo no terminal (P.13)

## FIRST



### FIRST(x):

- ① símbolos terminales de la regla
- ② Si tengo  $A \rightarrow BC_1$ , hacer el FIRST(B). Solo si B tiene:  $B \rightarrow d^* \lambda$ , como tiene  $\lambda$  vuelvo a hacer el first de C. Si el FIRST(C) NO deriva  $\lambda$ , paro. Si  $C \rightarrow \lambda$ , paso a h.

- Si en todos los elementos de la regla implican  $\lambda$  añado  $\lambda$  al FIRST(x). Cuando tenga uno que no implique  $\lambda$  entonces paro y NO pongo  $\lambda$  en la cadena de símbolos terminales.

$$\text{First}(S) = \{a, b, h, c, d, \lambda\}$$

$$\text{First}(T) = \{a, b, h\}$$

$$\text{First}(V) = \{c, \lambda\}$$

$$\text{First}(Z) = \{d, \lambda\}$$

## Follow

$$S \rightarrow TV^* VZ$$

$$T \rightarrow aT \mid bT \mid h$$

$$V \rightarrow \lambda \mid c \mid Z \mid h$$

$$Z \rightarrow \lambda \mid d \mid Z$$

- ① Buscar T en las consecuentes de las reglas

- Si detecto tengo un terminal, el follow es ese terminal  $S \rightarrow T \underline{C}$

- Si detecto tengo un no terminal,

Todos los elem. no nulos del first  $\xrightarrow{\text{hago el first de ese terminal}}$   $S \rightarrow T \underline{V} ; \text{Follow}(T) = \{c, \$\}$   
 $\xrightarrow{\text{(menos T) se ordenan}}$

- Si tengo  $A \rightarrow VT$ , hago el follow de A  $\xrightarrow{\text{as que voy a}}$   $\xrightarrow{\text{y hago follow(A)}}$   $\xrightarrow{\text{que es \$}}$

- Si A es el axioma, tiene \$

$$\text{Follow}(T) = \{c, \$\}$$

$$\text{Follow}(V) = \{d, \$\}$$

$$\xrightarrow{\text{axioma}} \text{Follow}(S) = \{\$\}$$

$$\text{Follow}(Z) = \{\$, h\}$$

## Condición LL(1)

$A \rightarrow \alpha | B | \gamma$  (siendo  $\alpha, B, \gamma \in (NUT)^*$ )

b)  $\text{First}(\alpha) \cap \text{First}(B) = \emptyset$

c) Si  $B \Rightarrow \lambda$ , si alguno de los conjuntos FIRST anteriores tiene  $\lambda$   
 $\text{First}(\alpha) \cap \text{Follow}(A) = \emptyset$

- \* Nunca  $\lambda$  puede estar en Follow (porque aparecerá \$ detrás de la cadena)
- \* Si S es el axioma, detrás de él sólo podremos poner el símbolo de final de cadena \$
- \* Follow(S), tenemos que ver en los consecuentes que aparece S
- \* FIRST(S) ( $\Rightarrow S \rightarrow TU$ )  $\Rightarrow \text{First}(S) = \text{FIRST}(T) + \text{FIRST}(U)$
- \* En el catálogo de FIRST, si todo el consecuente puede ser  $\lambda$  se anota  $\lambda$  al First (antecedente).

## Ejemplo: Comprobación de la condición LL(1)

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE'| \lambda \\ T &\rightarrow FT' \\ T' &\rightarrow *FT'| \lambda \\ F &\rightarrow (\varepsilon) \text{ lid} \end{aligned}$$

Vemos las sentencias que tienen más de un consecuente y resolvemos por pares

①  $E' \rightarrow +TE'| \lambda$

$\text{First}(+TE') \cap \text{First}(\lambda) = \{+\} \cap \{\lambda\} = \emptyset$

$\text{First}(+TE') \cap \text{Follow}(E') = \{+\} \cap \{\}, \$\} = \emptyset$

• Follow(E')

$E \rightarrow TE'$  (todo lo que tenga follow( $\varepsilon$ )  $\leq$  Follow( $E'$ )) axioma

$E' \rightarrow +TE' \rightarrow \text{Follow}(E') \leq \text{Follow}(E)$   $\text{follow}(\varepsilon) = \{\$, +\}$

$F \rightarrow (E)$

②  $T' \rightarrow *FT'| \lambda$

$\text{First}(*FT') \cap \text{First}(\lambda) = \emptyset = \{* \} \cap \{\lambda\}$

$\text{First}(*FT') \cap \text{Follow}(T') = \{* \} \cap \{ \}$

• Follow(T')

$\text{Follow}(\varepsilon) = \{ \}, \$ \}$

$T \rightarrow FT' (\text{Follow}(T) \leq \text{Follow}(T'))$

$\hookrightarrow \text{Follow}(T)$

$E \rightarrow TE'$

$E' \rightarrow +TE'$

$\text{Follow}(E') \leq \text{Follow}(T) \Rightarrow \{+, \}, \$ \}$

$\text{Follow}(E') \leq \text{Follow}(T) \Rightarrow \{+, \}, \$ \}$

$\text{Follow}(E') \leq \text{Follow}(T) \Rightarrow \{+, \}, \$ \}$

$T' \rightarrow +FT' (\text{Follow}(T) \leq \text{Follow}(T')) \rightarrow \text{no se anota nada más}$

## Análisis sintáctico descendente predictivo con Gramáticas LL(1)

→ Gramática LL(1) para hacer analizador sintáctico descendente

Gramáticas no válidas:

→ recursiva por la izq → para ningún descendiente

→ ambigua → para ningún sintáctico

→ No factorizada → No cumple la condición LL(1)

    ↳ símbolo no terminal con varias reglas y las consecuentes empiezan de la misma forma      $A \rightarrow \underline{\alpha} B_1 | \underline{\alpha} B_2 | \dots | \underline{\alpha} B_n$

    ↳ Transformar la gramática en no factorizada

$$A \rightarrow \underline{\alpha} A' | \gamma$$

$$A' \rightarrow B_1 | B_2 | \dots | B_n$$

### Ejemplo

$$S \rightarrow id := \bar{E} + E$$

$$S \rightarrow id := \bar{E} * E$$

$$\bar{E} \rightarrow (\bar{E})$$

$$\bar{E} \rightarrow id$$

Atípica es LL(1)

$$S \rightarrow id := \bar{E} S'$$

$$S' \rightarrow * \bar{E}$$

$$S' \rightarrow + \bar{E}$$

$$\bar{E} \rightarrow (\bar{E})$$

$$\bar{E} \rightarrow id$$

factorizar

### ① Analizador sintáctico descendente recursivo predictivo

- Una función para cada símbolo no terminal y una rama para cada regla.

    → If then else anidados

    → El token recibido determina qué rama se ejecuta y se recorre el consecuente de la regla. Para cada consecuente:

        → No terminal: se llama a su función

        → Terminal: se equipara. Si coincide con el token recibido se pide un token nuevo a An. Léxico. Si no coincide, error

sintáctico

- Las funciones se llaman recursivamente

- Main analizador sintáctico:

    → An. Sintáctico pide token a An. Léxico

    → Al terminar la ejecución de la función, si la cadena se ha leído entero el An. Sintáctico termina con éxito, sino error

- ② Analizador sintáctico descendente predictivo con tabla
- Gramática CCS.
  - Utilice una pila para construir el árbol de análisis de la cadena
  - La regla a aplicar en cada instante está almacenada en una tabla.
    - Fila: símbolos no terminales → mayúscula
    - Columna: símbolos terminales, \$ → minúscula
  - Funcionamiento: ver el símbolo de la cima de la pila y el siguiente token del an. léxico.
    - Cima de la pila terminal → minúscula
      - Debe coincidir con el mandado por el léxico. → equiparar, sacar terminal de la pila y Pedir sig token
    - Cima de la pila no terminal → mayúscula
      - Hay que aplicar una regla que se busca en la tabla en la fila del no terminal y la columna del token que pide An. Léxico. A → BCD
        - Sacar antecedente de la pila (A)
        - Meter consecuente en la pila →
      - ④ Funciona POR DERIVACIONES o la izq A → BCD
        - sig. símbolo, por eso se pone en la cima
  - Celda vacía: No hay regla para aplicar. Si ocurre hay error.
  - ⑤ La tabla tiene en cada casilla como máximo 3 regla
  - ⑥ Celda con más de una regla → NO es CCS.
  - ⑦ Celda vacía: caso de error
  - Construir la tabla A → ② → cadena símbolos terminales y no terminales
    - Para cada terminal  $\alpha$  de  $\text{FIRST}(\alpha)$ , añadir  $A \rightarrow \alpha$  a una celda
      - En la fila de A y en las columnas pertenecientes al  $\text{FIRST}(\alpha)$
    - Si en vez de un terminal tengo  $\lambda$  ( $\text{FIRST}(\lambda) = \lambda$ ), entonces
      - En la fila de A y en las columnas de  $\text{Follow}(A)$
      - Si \$ está en el  $\text{Follow}(A)$ , entonces se añade la regla en la fila de A y en la columna \$



Ver mis op.

Continúa de



405416\_arts\_esce\_ues2016juni.pdf

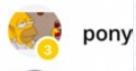
Top de tu gr.



7CR



Rocío



pony



Inicio

Asign.

**Ejemplo** construcción tabla de análisis

- $E \rightarrow TE'$

$$\text{FIRST}(T) = \{(), \text{id}\}$$

$$\text{FIRST}(FT') = \{(), \text{id}\}$$

- $E' \rightarrow +TE' \rightarrow \text{Fila } z^1, \text{ columna } +$

$$\text{FIRST}(+TE') = \{+\}$$

- $E' \rightarrow \lambda$

$$\text{FOLLOW}(E') = \{(), \$\}$$

- $T \rightarrow FT'$

$$\text{FIRST}(FT') = \{(), \text{id}\}$$

- $T' \rightarrow *FT'$

$$\text{FIRST}(*FT') = \{\omega\}$$

- $T' \rightarrow \lambda$

$$\text{FOLLOW}(T') = \{+, \$, ()\}$$

- $F \rightarrow (E) = \{(\}$

- $F \rightarrow \text{id} = \{\text{id}\}$

1.  $E \rightarrow TE'$
2.  $E' \rightarrow +TE'$
3.  $E' \rightarrow \lambda$
4.  $T \rightarrow FT'$
5.  $T' \rightarrow *FT'$
6.  $T' \rightarrow \lambda$
7.  $F \rightarrow (E)$
8.  $F \rightarrow \text{id}$

Tabla M		id	+	*	(	)	\$
$\bar{E}$	$\bar{E} \rightarrow TE'$	$\bar{E} \rightarrow TE'$					$E \rightarrow E'$
$\bar{T}'$			$E' \rightarrow *Tz'$				$E' \rightarrow E'$
$T$	$T \rightarrow FT'$						$T \rightarrow FT'$
$T'$			$T' \rightarrow *$	$T' \rightarrow FT'$	$T' \rightarrow *$	$T' \rightarrow *$	$T' \rightarrow T'$
$F$	$F \rightarrow id$						$F \rightarrow (E)$

④ Gramática NO LL: tengo más de una regla en una celda

$$S \rightarrow ABC$$

$$B \rightarrow b / \lambda$$

$$C \rightarrow c$$

$$\text{Follow}(A) = \{b, c\}$$

$$S \rightarrow ABC$$

$$B \rightarrow b / \lambda$$

$$C \rightarrow c / \lambda$$

$$\text{Follow}(A) = \{b, c, \$\}$$

↳ follow s

El FIRST de lo anterior que ve detrás de mi símbolo debe ser  $\lambda$  para hacer FOLLOW (anterior)

### ③ Analizador sintáctico ascendente con gramáticos LR(1)

- Con un solo token; el siguiente que el léxico manda, el sintáctico sabe lo que tiene que hacer.
- El árbol lo construye por reducción - desplazamiento
  - Si puede aplicar regla, reduce.
  - Si no puede aplicar una regla, desplaza.
- El analizador consulta siguiente token estado de encima de la pila
- El estado es imprescindible meterlo en la pila. (símbolos gramaticales no son necesarios, pero los metemos por clarificación)
- Acciones
  - desplazar → meter el token que manda el léxico en la pila
  - reducir → aplicar una regla → sacar en la pila lo que coincide con el consecuente de la regla y meter el antecedente.
  - aceptar → termina con éxito
  - error

- Estado: lo indica GOTO. Configuración de la pila
  - Inicial: estado inicial; la pila contiene \$0
  - Final: estado inicial \$0, el axioma introducido por la gramática y un estado

Método de construcción de las tablas acción y GOTO de un LR(1)

Tabla → fila: estado

→ columna: símbolos gramaticales y \$

Tabla acción: contiene la acción a realizar para cada par "estado", "símbolo".

Tabla GOTO: contiene el estado a apilar tras desplazar o reducir

• Tabla acción:

- Columnas: símbolos terminales y goto
- Filas: nº de estados

• Tabla GOTO:

- Columnas: las mismas que en acción y anade columnas para No terminales menos el axioma
- Filas: nº de estados

En este tabla pongo en los no terminales el estado a apilar

		i0	+	*		
		d5			GOTO	
Acción	0	d5			1	T/F
	1		d4		8	
2	Acep		r2			

d# : desplazar y apilar el estado #

r# : reducir por la regla #

Acep: aceptar

desplazar el token

i0 de la entrada a la pila + apilar el estado 5

→ aplicar la regla 2

"reducir por la regla 2"

#: estado a apilar (se necesita después de reducir).

④ LR(1) si la celda solo tiene una acción.

### ④ Desplazamiento

- Apilar token
- Apilar estado
- Llamar léxico

### ④ Reducción

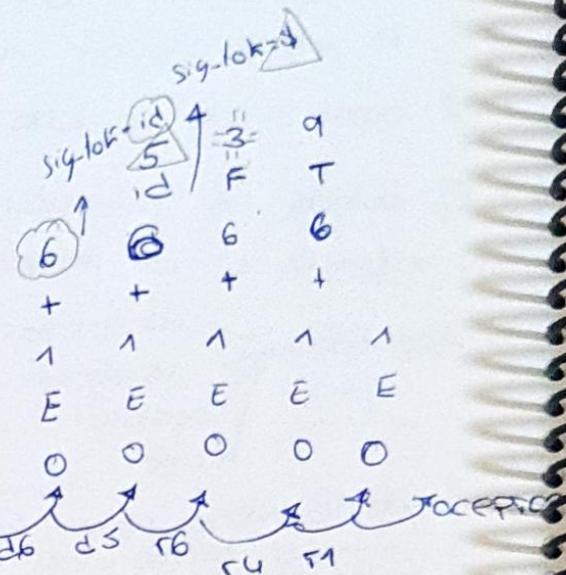
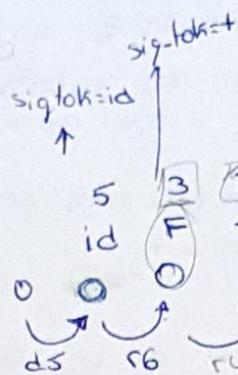
- Sacar consecutiva
- meter antecedente
- Mirar el Goto para apilar el estado

Ejemplo D 17 tengo la tabla y la Gramática

PILA y TABLA

id + id \$

0.  $E' \rightarrow E$
1.  $E \rightarrow i + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (i)$
6.  $F \rightarrow id$



ACCION

	id	+	*	(	)	\$	GO TO
0	d5			d4			E T F
1		d6				acept.	1 2 3
2		r2	d7		r2	r2	
3		r4	r4		r4	r4	
4	d5		d4				8 2 = 3 =
5		r6	r6	r6		r6	
6	r5		d4				9 3
7	d5		d4				10
8		d6		d11			
9		r1	d7		r1	r1	
10		r3	r3		r3	r3	
11		r5	r5	r5	r5		
12							

Acción 5 con + no reducir por la regla 6.

- 1) Sacar de la pila 2 elementos
- 2) Apilar en el 0 encima el antecedente F
- 3) Mirar en el Goto OF que me da el estado 0
- 4) Mirar 3 con + → r4

3  
F  
0

## → Cómo construir la tabla LR(1)

→ ① Gramática aumentada: añadir un nuevo axioma que tenga una resta que vaya al axioma antiguo  $S' \rightarrow S$

→ ② Hemos: regla gramatical con punto en alguna posición del consecuente.  
El punto separa la parte del consecuente que ya se ha recibido de la que se espera recibir para aplicar la reducción

Todos los ítem  
son PUNTO al  
final menos  
 $S' \rightarrow S$ , indican  
reducción

Regla Gramatical

$A \rightarrow Xyz$

Ítems posibles

$A \rightarrow .xyz$  = llega t2 punto reduce

$A \rightarrow x.yz$  indica que el símbolo t1 se ha recibido (todas las simbolas necesarias para aplicar la regla)

$A \rightarrow xy.z$  indica que el símbolo t2 se ha recibido (todas las simbolas necesarias para aplicar la regla)

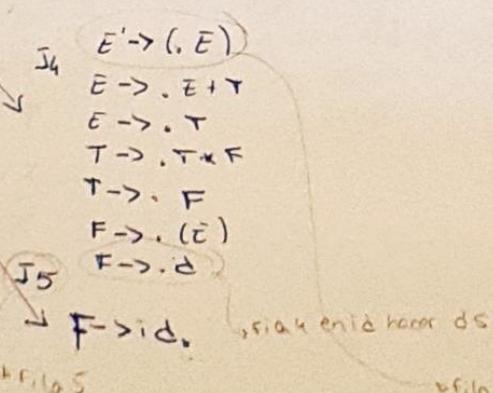
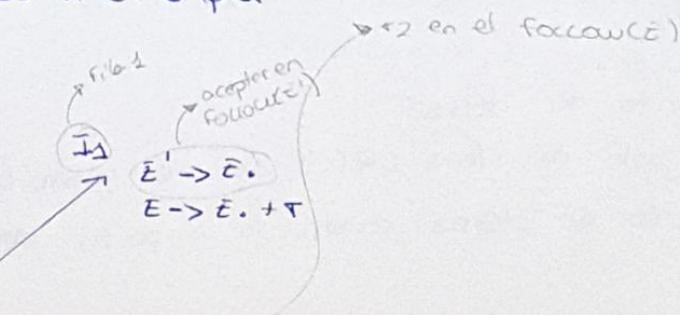
$A \rightarrow .$

$A \rightarrow \lambda$   
No saco nada de la pila  
y meto A en la pila

0.  $E' \rightarrow E$
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow \tau * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$

7.  $E' \rightarrow .E$
8.  $E \rightarrow .E + T$
9.  $E \rightarrow .T$
10.  $T \rightarrow .\tau * F$
11.  $T \rightarrow .F$
12.  $F \rightarrow .(E)$
13.  $F \rightarrow .id$

$\alpha \rightarrow \alpha + B$   
desplazar +  
 $\alpha \rightarrow \alpha.$   
dejar para  
follow(A)



$Follow(T) = \{+, *, (, )\}$  primera tabla segund  
 $Follow(E) = \{+, *\}$   
 $Follow(F) = *$



[Ver mis op](#)

Continúa d



405416\_arts\_esce  
ues2016juniy.pdf

Top de tu g

- 7CR
- Rocío
- pony
- Inicio
- Asign.

# Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the  
App Store

GET IT ON  
Google Play

Conflicto: en un mismo estado, diferentes ítems válidos de ese estado me indican diferentes acciones.

① Reducción / reducción: en una casilla puedo aplicar dos

o más reglas distintas

si  $\text{follow}(A)$  y  $\text{follow}(B)$  son ✓ no hay problema

$A \rightarrow \alpha$ . → Reduce por  $A \rightarrow \alpha$  para  $\text{follow}(A)$

$B \rightarrow \beta$ . → Reduce por  $B \rightarrow \beta$  para  $\text{follow}(B)$

② Reducción / desplazamiento: un ítem dice reduce y otro desplaza

$A \rightarrow \alpha$ . → Reduce por  $A$

$B \rightarrow \beta$ . t ↳ Desplazar t ] Conflicto si t pertenece

al  $\text{follow}(A)$

Solo estudiar los que tengan reducción.

$E' \rightarrow E$ .

$E \rightarrow E + T$  ↳ Tengo que ver que + no esté en  $\text{follow}(E')$

$E \rightarrow T$ .

$T \rightarrow T * F$  ↳ mirar que \* no esté en  $\text{follow}(E)$

→ ③ Cierre de un conjunto de ítems:

Si  $I$  es un conjunto de ítems ( $R(0)$ ) de una gramática  $G$ , entonces  $\text{cierre}(I)$  es el conjunto de ítems construido a partir de  $I$  mediante las dos reglas:

1) Inicialmente, todos los elementos de  $I$  se ordenan a cierre ( $I$ )

2) Cierre( $I$ ) añade los reglos del no terminal con. al principio

Ejemplo

1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow . id$

$$I = \{F \rightarrow (. E)\}$$

$$\text{cierre}(I) = \{F \rightarrow (. E)\}$$

como espero cosa  
que encajen con  $E$ ,  
meto  $E$  y sus  
reglas

$$\textcircled{1} \quad E \rightarrow . E + T$$

$$E \rightarrow . T$$

$$\textcircled{2} \quad T \rightarrow . T * F$$

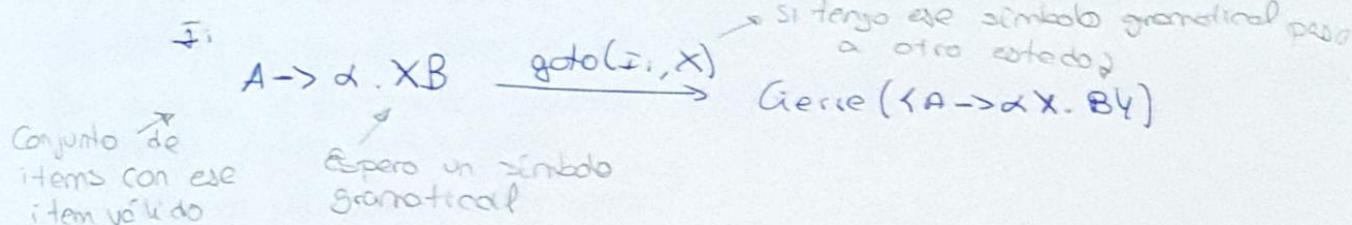
$$T \rightarrow . F$$

$$\textcircled{3} \quad F \rightarrow . (E)$$

$$F \rightarrow . id$$

4

→ ④ Goto de un conjunto de ítems y un símbolo gramatical



$$\mathcal{I}_4 = \{ F \rightarrow (\cdot E) \quad \text{goto } \mathcal{I}_4, \varepsilon \} = \text{Cierre}(F \rightarrow (\bar{E} \cdot), E \rightarrow E \cdot + T)$$
$$T \rightarrow \cdot E + T \quad = \{ F \rightarrow (\bar{E} \cdot) \}$$
$$E \rightarrow \cdot T$$
$$T \rightarrow \cdot T * F$$
$$T \rightarrow \cdot F$$
$$F \rightarrow \cdot (\bar{E})$$
$$F \rightarrow \cdot id$$

→ ⑤ Construcción de la colección canónica de ítems LR(0)

Formada por los estados  
de ítems.

## LL1

- Cada regla solo puede encontrarse en la fila de su no terminal
- La tabla no puede tener una fila vacía
- Una misma regla puede aparecer únicamente en una fila de la tabla y podría estar varias veces.
- La columna \$ puede estar vacía
- Para cada símbolo no terminal que tiene más de una regla  $A \rightarrow \alpha \mid \beta$ ;  $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$

$$A \rightarrow \alpha \mid \beta; \text{First}(\alpha) \cap \text{Follow}(A) = \emptyset$$

Para que sepa qué regla debe aplicar  
Si tengo en las dos reglas el mismo símbolo no se sabe aplicar

- Solo se aplica una regla en cada caso

- No se puede retroceder.

- Gramática L1

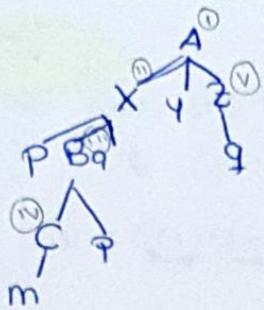
- No gramática recursiva por lo que  $(A \rightarrow A\beta)$  → Para ningún descendiente
- No gramática amigable → construir dos árboles distintos para una misma cadena de entrada
- No gramática no factorizada →  $(A \rightarrow \underline{id} * E)$  Simbolo no terminal con varios reglas y los consecuentes empiezan de la misma forma
- Todas las reglas del axioma de la gramática estarán cobocadas en la fila del axioma

## LR1

- El árbol se construye por reducción / desplazamiento
  - Aplicar regla = reducir
  - No aplicar regla = desplaza
- Analizador consulta el estado encima de la pila siguiente token
- Tabla ← Fila: estado
- Columna: símbolos gramaticales, \$
- Gramática aumentada se obtiene a partir de la gramática original, añadiendo un nuevo axioma y una nueva regla que deriva el nuevo axioma de la gram. original
- La gramática aumentada es necesaria para que An. Sintáctico Descendente LR sea capaz de usar la acción aceptar
- La gramática aumentada sirve para asegurarse que, al reducir por la nueva regla de esta gramática, se aceptará la cadena de entrada.
- En la gramática aumentada al nuevo axioma nunca aparece en la parte derecha de la regla
- No puede haber columnas vacías
- Desplazamiento: significa que se desplaza el terminal de la entrada a la pila

Analizador sintáctico descendente  $\rightarrow$  crear el árbol de la raíz a los nodos

- 1) No puede ser recursiva por la regla ( $A \rightarrow A\alpha$ )
- 2) Coger el símbolo no terminal más a la izq y aplicar su regla hasta que todos los nodos sean hojas.



Solo hay una regla a aplicar

$$\begin{cases} 1 A \rightarrow Xyz \\ 2 X \rightarrow pBq \\ 3 B \rightarrow q \\ 4 B \rightarrow CP \\ 5 C \rightarrow m \end{cases}$$

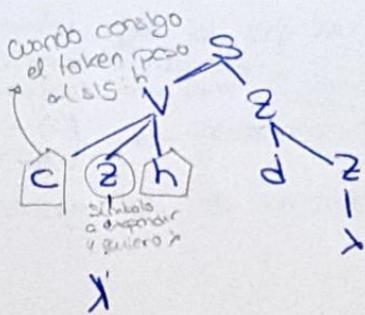
Parse: 1, 2, 4, 5, 3

Hay varias reglas, ver cuál de ellas me lleva a conseguir el siguiente token

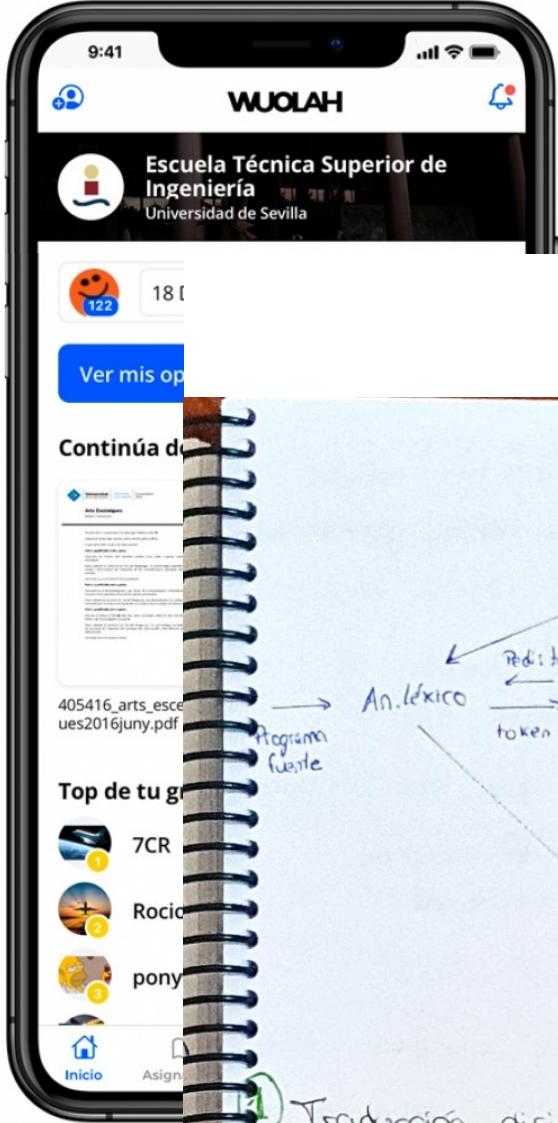
- 1  $S \rightarrow TV \mid VZ$
- 2  $T \rightarrow aT \mid bT \mid h$
- 3  $V \rightarrow \lambda \mid cZ \mid h$
- 4  $Z \rightarrow \lambda \mid dZ$

Tokens:  $c \quad hd$   
 $TV \quad no$   
me lleva  
 $a \quad c$

Parse: 2, 3



Funciona por derivaciones hacia izquierda



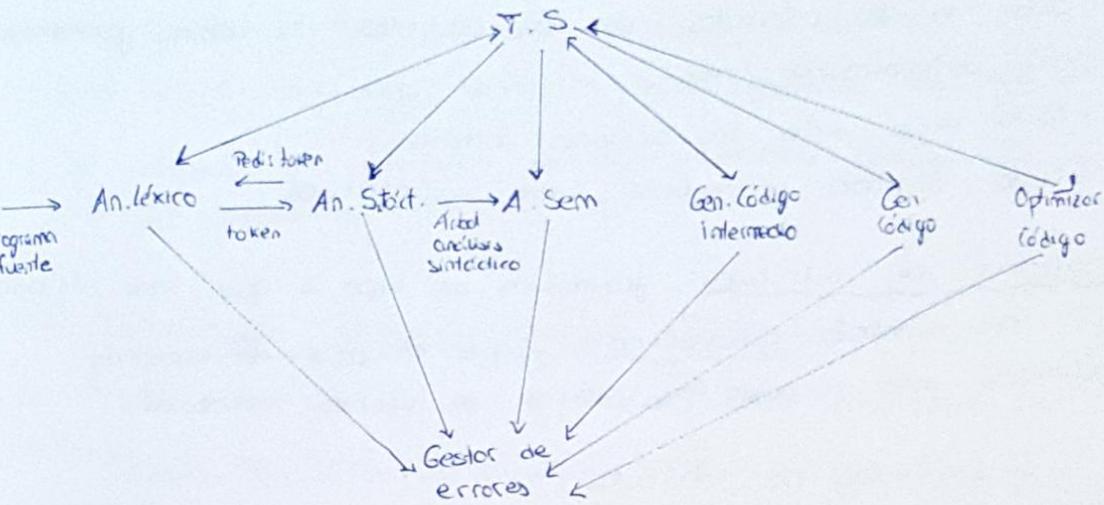
# Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the  
App Store

GET IT ON  
Google Play

## Analizador semántico



### 1 Traducción dirigida por la sintaxis

- Analizador Sintáctico invoca las rutinas que realizan el Análisis Semántico y la Generación de Código Intermedio.
- An. Sintáctico comunica o An. Semántico la regla que se ha utilizado para construir un fragmento de árbol. Ese fragmento es analizado por el An. Semántico.
- Una vez hecho la traducción dirigida por la sintaxis se obtiene un árbol de análisis sintáctico anotado con info semántica adicional sobre cada nodo del árbol de análisis Sintáctico.
- La traducción dirigida por la sintaxis se basa en gramáticas de atributos.

### 2 Gramáticas de atributos

Atributo permite manejar información semántica asociada a los símbolos de la gramática. Esta información asociada son los tipos de la gramática. Cada símbolo gramatical tendrá sus atributos:

Notación: Símbolo. Atributo

Expresión tipo  
 Identificador. Lugar  
 Clé-entera. valor  
 Sentencia. código

Acciones o rutinas semánticas: operaciones que permiten calcular el valor de los atributos de los distintos símbolos gramaticales de una determinada regla.

- Cada regla tiene sus acciones semánticas
- Cada símbolo gramatical tiene sus atributos.

Gramática de atributos: gramática de tipo 2 que tiene atributos para los símbolos gramaticales y que dispone de acciones semánticas para calcular los valores de dichos atributos.

### 2.1 Tipos de atributos. → info adicional de algún símbolo

(Práctica) **Sintetizados**: su valor en un nodo del árbol de análisis sintáctico se calcula a partir de los valores de los atributos de las hijas de dicho nodo.

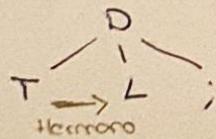
regla:  $E \rightarrow (E_1)$

↑ subíndices  
para diferenciar  
los mismos símbolos  
gramaticales que pueden aparecer  
repetidos en una regla ( $E$  aparece 2 veces)

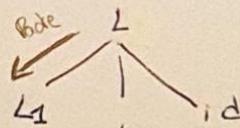
$E.\text{tipo} := E_1.\text{tipo}$

**Heredados**: su valor en un nodo del árbol se calcula como una función de valores de los atributos del padre e hermanos de dicho nodo.

$D \rightarrow T L$   $L.\text{tipo} := T.\text{tipo}$



$L \rightarrow L_1, id$   $L_1.\text{tipo} := L.\text{tipo}$



## 2.2 Acciones semánticas

- Evaluar los atributos de una gramática de atributos
- Cada acción o rutina semántica está ligada a una regla semántica.
- Se emplean los atributos de los símbolos gramaticales de su regla sintáctica

### ③ Traducción Dirigida por la Sintaxis

La TDS es la técnica que se utiliza tanto para hacer la comprobación de tipos como para construir el generador de código intermedio. Consiste en definir atributos para los elementos gramaticales de la GCL y asociar reglas semánticas a las reglas sintácticas de manera que permitan calcular el valor de un atributo en función de los atributos de los símbolos de la regla.  
→ A cada regla sintáctica se le asocia una regla semántica.

#### 3.2. Definición Dirigida por la sintaxis (DDS)

No se dan detalles. Únicamente se indica la regla semántica correspondiente a cada regla sintáctica, sin decir en qué momento se aplica la regla semántica.

#### Rutinas semánticas

- Cada rutina semántica está ligado a una regla sintáctica concreta, y solo puede hacer referencia a los símbolos de dicha regla.
- En esa regla sintáctica vamos a utilizar los atributos de los símbolos gramaticales que aparecen en esa regla sintáctica
- Las rutinas semánticas en una DDS indican qué hay que hacer para esa regla sintáctica.

#### Ejemplos rutinas semánticas

$$\begin{array}{ll} E \rightarrow (E) & | \quad E.\text{tipo} = E_1.\text{tipo} \\ L \rightarrow L_1, id & | \quad L_1.\text{tipo} = L.\text{tipo} \\ & \quad \text{Regla semántica} \end{array}$$

## Ejemplos DDS

1.  $D \rightarrow TL;$

2.  $T \rightarrow int$

3.  $T \rightarrow float$

$L.tipo := T.tipo$   $\rightsquigarrow$  atributo heredado

$T.tipo := ento$   $\rightsquigarrow$  atributo sintetizado

$T.tipo := real$

lista  $\left\{ \begin{array}{l} L \rightarrow L, id \\ Q \rightarrow id \\ \text{(lista de id separada por comas)} \end{array} \right.$

$L.tipo := L.tipo$

AñadeTipoTS(id.pos, L.tipo)

AñadeTipoTS(id.pos, L.tipo)

## Gramática

1. Conocer el significado de la gramática. Que debe hacer an. semántico

2. Declaración de variables: asociar a una variable el tipo

2.  $T \rightarrow int$   $\rightsquigarrow T.tipo := entero$

3.  $T \rightarrow float$   $\rightsquigarrow T.tipo := real$

5.  $L \rightarrow id$   $\rightsquigarrow$  An Semántica tiene que obtener el tipo con el que se declara esa variable y meterlo en la TS. Para ello necesito saber la posición de la TS introducir esa info  
 $\hookrightarrow id.pos \rightsquigarrow$  refleja la posición del id en la TS

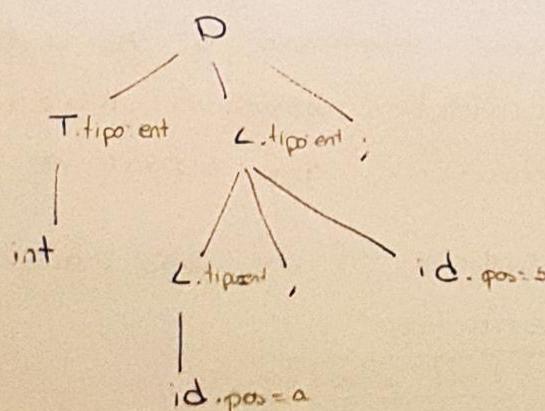
4.  $L \rightarrow L, id$   $\rightsquigarrow$

1.  $D \rightarrow TL$   $\rightsquigarrow$  En las reglas de T obtendremos el tipo de T

En las reglas de L usamos el tipo L

Asignar valor a L.tipo.  $\rightsquigarrow L.tipo = T.tipo$

int a, b;



El analizador sintáctico y el analizador semántico se pueden hacer simultáneamente cuando se utilizan atributos que se calculan en función de atributos que fluyen de izquierdo a derecho. Cuando se utilizan atributos de lo derecho podemos obligando a hacer primero el sintáctico y luego el semántico.



# Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the  
App Store

GET IT ON  
Google Play



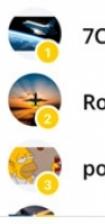
18

Ver mis op

Continúa d



Top de tu gr



Rocio

pony



Asign.

### 3.3.3 Definiciones con Atributos por la izquierda

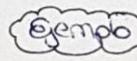
- Permiten evaluar reglas semánticas al tiempo que el Analizador Sintáctico usa una regla sintáctica.

• Una Definición Dirigida por la Sintaxis es una Definición con Atributos por la izquierda si cada atributo heredado  $X_j$ , para  $j$  con valores entre 1 y n, que pertenece al lado derecho de la regla  $A \rightarrow x_1, x_2, \dots, x_n$  depende de:

- Atributos de los símbolos  $x_1, x_2, \dots, x_{j-1}$  (a la izq de  $x_i$  en el consecuente)
- Los atributos de  $A$  no heredados del padre

### 3.2 Esquema de Traducción (EDT)

- Las acciones (reglas semánticas) se intercalan entre los consecuentes de las reglas.
- En el EDT se muestra el orden en que se ejecutan las acciones semánticas y en la DDS no. (Qué hay que hacer y cuándo)  $\rightarrow$  EDT  $\rightarrow$  DDS, solo qué hay que hacer
- Se ponen entre {} las acciones semánticas.



$\rightarrow$  EDT y DDS indican qué hay que hacer  $\rightarrow$  mismas acciones semánticas. EDT indica el orden

$D \rightarrow TL$

$T \rightarrow int$

$T \rightarrow float$

$L \rightarrow L, id$

$L \rightarrow id$

$D \rightarrow T \{ L.tipo = T.tipo \} L;$

$T \rightarrow int \{ T.tipo = ent \}$

$T \rightarrow float \{ T.tipo = real \}$

$L \rightarrow \{ L.tipo = L.tipo \} L, id \{ AnadeTipos(id.pas, L.tipo) \}$

$L \rightarrow id \{ AnadeTipos(id.pas, L.tipo) \}$

### 3.2.1 Diseño de un EDT

- Si solo hay atributos sintetizados, puede construirse el EDT colocando las acciones semánticas al final del lado derecho de la regla. (Al final de todas las símbolos gramaticales)

$\Rightarrow E \rightarrow (E_1) \{ E.tipo = E_3.tipo \}$

WUOLAH

- Si hay atributos sintetizados y heredados de cumplirse:
  - 1) Un atributo heredado para un símbolo que está en la parte derecha de la regla se debe calcular en una acción semántica colocada antes de dicho símbolo.
 

⇒  $S \rightarrow \{ A_3.h = 3; A_2.h = 24 \} A_3 A_2$  → bien  
 $S \rightarrow A_1 A_2 \{ A_3.h = 3; A_2.h = 24 \}$  → mal
  - 2) Una acción semántica no debe referir a un atributo sintetizado de un símbolo situado a la derecha de dicha acción.
  - 3) Un atributo sintetizado para el no terminal de la izquierda solo se puede calcular después de que se hayan calculado todos los atributos a los que hace referencia.
 

$L \rightarrow id:T \{ L.t := id.t; id.t := T.t \}$  → No puedo hacer  
 $L \rightarrow id:T \{ id.t := T.t; L.t := id.t \}$  →  $L.t = id.t$  si no se qué valor tiene id.t

#### ④ Comprobación de tipos

En las reglas (acciones semánticas) hay que

- Comprobar que los tipos de la expresión son correctos
- Generar el código intermedio correspondiente

- i) Determinar los tipos a manejar en el analizador semántico

Sistema de tipos: serie de normas para asignar expresiones de tipos a las distintas partes de un programa. ⇒ Un comprobador de tipos implementa un sistema de tipos.

Expresiones de tipos: denotan el tipo de una construcción de un lenguaje.

El comprobador de tipos implementa un sistema de tipos. Tiene como función asegurar que el tipo de una construcción coincide con el previsto en el contexto. Por ejemplo, que se sumen elementos del mismo tipo o tipos compatibles, no una matriz y una función.

## 1.1 Expresiones de tipos

### ① Tipos básicos

Tipos básicos	- entero - real - lógico - carácter - cadena	(int) (float) (boolean) (char) (string)
del lenguaje fuente		

- tipo-ok: se usará cuando una construcción no tiene un tipo concreto pero queremos denotar que los tipos de esa construcción son correctos ( $\rightarrow$  no tiene tipo if, while)
- tipo-error: ejemplo, producto de entero y cadena.  
if con una condición de tipo cadena
- vacío: denota ausencia del tipo. Lo usaremos cuando una función no devuelva nada, o cuando una función no tiene parámetros

### ② Nombres de tipo: nombres que el programador define en el programa fuente para declarar un determinado tipo de datos (tipo de datos fib) $\rightarrow$ estructura de datos

### ③ Construcciones de tipo: se basan en otros tipos para crear un nuevo tipo más elaborado

a) Vectores: si  $T$  es una expresión de tipo, entonces: array( $I, T$ )  
Ejemplo: var A: array [3...10] of integer  $\Rightarrow$  array(3..10, integer)

b) Productos: Si  $T_1$  y  $T_2$  son expresiones de tipo, entonces su producto cartesiano:  $T_1 \times T_2$

c) Registros: se utiliza la notación de producto cartesiano. La diferencia entre registro y producto es que los campos de un registro tienen nombres. Por tanto, el constructor de tipos "record" se aplica a una tupla formada por nombres de campos y tipos de campos.

Ejemplo: type fib = record

dirección: integer,

lexema: array [1..15] of char;

end record;

$\hookrightarrow$  record(dirección x integer) (lexema x array [1..15, char])

$\hookrightarrow$  record(nombre.campo1 x tipo.campo1) x ... x  
(nombre.campoN x tipo.campoN)

d) Punteros: si  $T$  es una expresión de tipo, entonces  $\text{pointer}(T)$  es una expresión de tipo que indica el tipo "puntero a un objeto tipo  $T$ ".

Ejemplo  $\text{verp} : \uparrow \text{file} \rightarrow \text{Pointer(file)}$

e) Funciones

$D \rightarrow R = \text{tipo-param-entada} \rightarrow \text{tipo-param-salida}$

Ejemplo:

- $\text{area\_suma}(x, y: \text{integer}) \rightarrow \text{return char}$   
 $(\text{integer} \times \text{integer}) \rightarrow \text{char}$
- $\text{functionf}(a, b: \text{char}) \rightarrow \text{return } \uparrow \text{integer}$   
 $(\text{char} \times \text{char}) \rightarrow \text{pointer(integer)}$

Ejemplo: declaraciones de un cencoraje de TIPO Pascal

Programa

$P \rightarrow \langle \text{desp.} = 0 \rangle$

$D; S \quad \downarrow 4$   
declaración sentencia

$D \rightarrow P; D \quad \downarrow 4$  → una declaración pueden ser varias seguidas

$D \rightarrow \text{id: } T \quad \{ \text{insertaTiTS(id.pos, T.tipo)} \rightarrow \text{asignar el tipo de } T \text{ a la posición de id}$   
Declaración individual       $\text{insertaDespTS(id.pos, desp)}$   
 $\text{desp} = \text{desp} + T.\text{ancho} \downarrow$

\*  $T \rightarrow \text{char} \quad \{ T.\text{tipo} = \text{car}, T.\text{ancho} = 1 \}$

$T \rightarrow \text{integer} \quad \{ T.\text{tipo} = \text{ent}, T.\text{ancho} = 2 \}$

$T \rightarrow \text{boolean} \quad \{ T.\text{tipo} = \text{logico}, T.\text{ancho} = 1 \}$

Puntero  $T \rightarrow \uparrow T_2 \quad \{ T.\text{tipo} = \text{pointer}(T_2.\text{tipo}), T.\text{ancho} = 4 \}$  → un puntero ocupa 4B

$T \rightarrow \text{array [num] of } T_1$

$\{ T.\text{tipo} = \text{array}(1..num.valor, T_1.\text{tipo}) \rightarrow \text{array desde 1 hasta el valor de num, y de tipo } T_1$   
 $T.\text{ancho} = \text{num.valor} * T_1.\text{ancho} \downarrow$

Como en las reglas de arriba teníamos  $T.\text{tipo}$  y  $T.\text{ancho}$ , tenemos que usar los dos en todas las reglas de  $T$

ancho lo que ocupa cada tipo de datos



# Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the  
App Store

GET IT ON  
Google Play



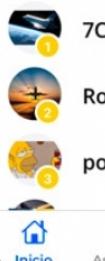
18

Ver mis op

Continúa de



Top de tu g



## 4.2 Sentencias de un lenguaje tipo Pascal

2º Asignación  
3º Sentencia condicional

- $S \rightarrow S_1; S_2 \rightarrow$  una sentencia puede ser un grupo de sentencias

• La sentencia es correcta cuando  $S_1$  y  $S_2$  son correctas. Si alguna de las dos no es correcta, será de tipo error.

$\{ S.\text{tipo} := \text{if } (S_1.\text{tipo} = \text{tipo-ok} \text{ and } S_2.\text{tipo} = \text{tipo-ok})$   
 $\quad \quad \quad \text{then tipo-ok}$   
 $\quad \quad \quad \text{else tipo-error} \}$

- $S \rightarrow \text{id} := E \rightarrow$  asignación

• Hay que definir el atributo  $S.\text{tipo}$ . Para que la sentencia sea correcta, el tipo del id y de la expresión debe coincidir.

$\{ \text{id}.\text{tipo} := \text{buscarTipoTS(id, pos)} \rightarrow$  buscar tipo de id  
 $S.\text{tipo} := \text{if } (\text{id}.\text{tipo} = \text{E}.\text{tipo})$   
 $\quad \quad \quad \text{then tipo-ok}$   
 $\quad \quad \quad \text{else tipo-error} \}$

- $S \rightarrow \text{if } (E) \text{ then } S_1 \rightarrow$  sentencia condicional

• Sentencia correcta cuando  $E$  sea de tipo lógico la sentencia  $S_1$  también sea correcta

$\{ S.\text{tipo} := \text{if } (E.\text{tipo} = \text{logico})$   
 $\quad \quad \quad \text{then } S_1.\text{tipo} \rightarrow$  si es correcta, el tipo  
 $\quad \quad \quad \text{else tipo-error} \}$  de  $S$  será igual al tipo de  $S_1$

Al heredado  
 $S_3.\text{atb} = S_4.\text{atb}$

PG 6 lorenz

Conversion de tipos: el comprobador de tipos tiene como misión verificar que la manipulación de tipos es correcta. La conversión de tipos se da cuando hay dos operadores de distinto tipo. Dos conversiones:

- Conversión implícita: la conversión se lleva a cabo en la acción semantics de la regla donde se realiza.
- Conversión explícita: funciona como una llamada a una función: recibe un tipo y devuelve otro.

... implícita:  $E_1 \rightarrow \text{opArit } E_2 \quad E_1.\text{tipo} = \text{if } (E_2.\text{tipo} = E_1.\text{tipo}) \text{ then } E_2.\text{tipo}$   
 $\quad \quad \quad \text{else error-tipo} \quad \quad \quad$

... si no hay coincid. Los elementos deben ser del mismo tipo o convertirse mediante funciones

entero + entero  $\rightarrow$  OK

entero + real  $\rightarrow$  intreal(entero) + real  
 real + entero  $\rightarrow$  real + intreal(entero)  $\rightarrow$  exacta

#### 4.3 Expresiones en un lenguaje tipo Pascal.

- $E \rightarrow \text{num}$  { $\bar{E}.\text{tipo} := \text{ent}$ }
- $E \rightarrow \text{true}$  { $\bar{E}.\text{tipo} := \text{lógico}$ }
- $E \rightarrow \text{id}$  { $\bar{E}.\text{tipo} := \text{buscaTipo(id, pos)}$ }
- $E \rightarrow E_1 + E_2 \rightarrow$  los tipos de ambos sumandos deben ser enteros  
 { $\bar{E}.\text{tipo} = \text{if } (\bar{E}_1.\text{tipo} = \text{ent} \text{ and } \bar{E}_2.\text{tipo} = \text{ent})$   
     then ent  
     else tipo\_error}
- $E \rightarrow \bar{E}_1 \text{ and } E_2 \rightarrow$  los dos operandos deben ser lógico  $\rightarrow$  resultado es lógico  
 { $\bar{E}.\text{tipo} = \text{if } (\bar{E}_1.\text{tipo} = \text{lógico} \text{ and } \bar{E}_2.\text{tipo} = \text{lógico})$   
     then lógico  
     else tipo\_error}
- $E \rightarrow \bar{E}_1^{\star} \rightarrow$  PUNTERO  
 $\bar{E}_1$  tiene que ser de tipo puntero. El tipo de la expresión viene dado por lo que apunta ese puntero. Tengo que averiguar si es puntero a entero, puntero a real...  $E$  será del tipo que apunte el puntero  
 { $\bar{E}.\text{tipo} = \text{if } (\bar{E}_1.\text{tipo} = \text{pointer}(t))$   
     then  $t$   
     else tipo\_error}
- $E \rightarrow \bar{E}_1[\bar{E}_2] \rightarrow$  acceso a un elemento de un vector  
 { $\bar{E}.\text{tipo} = \text{if } (\bar{E}_1.\text{tipo} = \text{array}(s, t) \text{ and } \bar{E}_2.\text{tipo} = \text{ent})$   
     then +  
     else tipo\_error}
 

índice de los vectores son enteros  
índice tipo de los elementos

$\bar{E}_1$  tiene que ser un vector y  $\bar{E}_2$  un entero. El tipo de  $\bar{E}$  es el tipo de los elementos del vector
- $E \rightarrow E_1 < E_2 \rightarrow$  operador relacional. Solo se comparan enteros  $\rightarrow$  resultado tipo lógico  
 { $\bar{E}.\text{tipo} = \text{if } (E_1.\text{tipo} = \text{ent} \text{ and } E_2.\text{tipo} = \text{ent})$   
     then lógico  
     else tipo\_error}
 

4 < 5?  $\rightarrow$

5 < 3?  $\rightarrow$

comprobación de tipos  
implementa un sistema  
de tipos. Tiene como  
función asegurar que el  
tipo de una construcción  
coincida con el previsto  
en el contexto

## Gestión de errores

- Detección de errores: localizar el error y notificarlo al usuario.
- Recuperación de errores: localizar el error y suponer que no se ha producido para continuar la búsqueda de nuevos errores.
- El compilador
  - Detecta errores de tipo ortográfico, sintáctico y semánticos
  - No detecta errores que vayan contra la lógica del programa
- Gestor de errores debe informar del tipo de error detectado y su localización.

Actuación ante un error:

- No advertir la presencia de un error
- Señalar el error y abandonar la compilación
- Señalar el error y continuar la búsqueda de nuevos errores
- Arreglar el error y proseguir la compilación

Tipos de errores

- Léxicos
- Sintácticos
- Semánticos
- De ejecución
- Del compilador

→ Errores léxicos: cuando no sea posible encontrar un token que equipare con alguno de los patrones del lenguaje, o cuando no se cumplen las restricciones impuestas por la definición del lenguaje fuente para los lexemas 

- longitud de id de más de 63 caract
- rango de constantes máx 16 bits

→ Errores sintácticos: cuando no es posible equiparar una regla Sintáctica con la secuencia de tokens recibidos (errores por ausencia de delimitadores ( $i:=5 ; j=8$ ), uso de palabras incorrectas if(condición) then a = f;