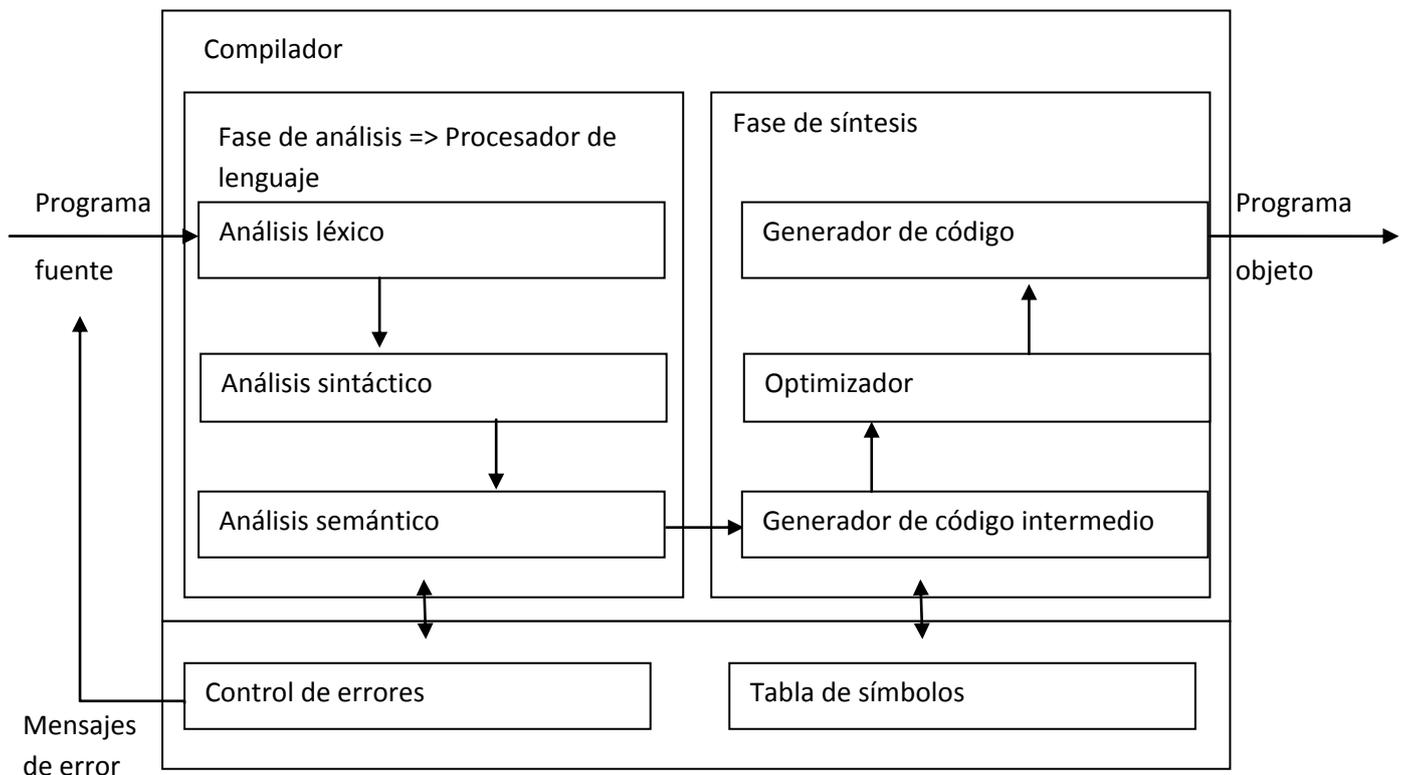


Procesadores del lenguaje

Pau Arlandis Martínez

Introducción

Un compilador es un programa que traduce un lenguaje (código fuente de otro programa) de código en otro texto en lenguaje objeto.



Este es el esquema básico de un compilador, veámoslo por partes:

Fase de análisis

En la fase de análisis se estudia el lenguaje del programa fuente, errores, elementos de interés... Esta fase también se denomina procesador del lenguaje y es lo que vamos a ver en

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

Análisis semántico

Se encarga de estudiar el significado de todos los elementos dentro del texto.

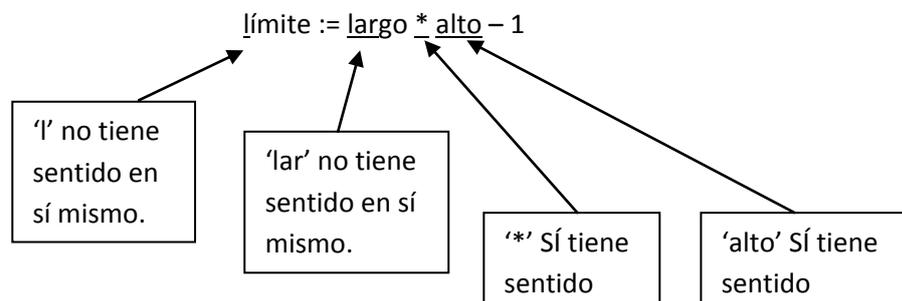
Ejemplo

Texto Fuente

límite := largo * alto - 1

Analizador léxico

El analizador léxico estudia carácter a carácter conociendo las estructuras mínimas que tienen sentido por sí mismas. A estas se las denomina *Tokens*.



Texto resultado

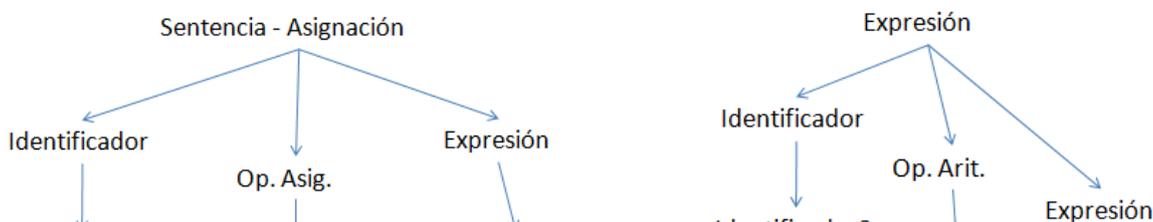
Identificador1/op.asignación1/identificador2/op.
aritmética1/identificador3/op.aritmético2/constante_entera1/

A grandes rasgos este sería el resultado del analizador. Cada *token* necesitaría más información pero eso lo veremos más adelante.

Analizador sintáctico

Estudia la estructura de la secuencia de *tokens* para conocer si es una estructura permitida o no lo es.

En el ejemplo:



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Control de errores

Cada fase de análisis tiene sus propios errores:

- **Errores léxicos:** Se utilizan operadores inexistentes o palabras mal formadas → *limi?te*, **/*
- **Errores sintácticos:** Se estructuran mal los *tokens*, de forma incorrecta o confusa → *id1 op.asig id2 cte_entera*
- **Errores semánticos:** La estructura de *tokens* elegida es correcta pero no tiene sentido → $id1^{(Entero)} op.asig expr^{(real)}$

Todos estos errores se analizan en el control de errores.

Tabla de símbolos

La tabla de símbolos es una colección de símbolos que utiliza nuestro texto para almacenar información de cada *token*. El procesador del lenguaje interactúa con ella para almacenar y consultar su contenido.

Estos 5 módulos de la fase de análisis son, básicamente, los temas y el contenido de la asignatura. Para completar la información, introduzcamos también el contenido de la fase de síntesis.

Fase de síntesis

Una vez resuelta sin errores la fase de análisis comienza la fase de síntesis que será la encargada de construir el texto objeto del texto fuente que hemos utilizado. La fase de síntesis se subdivide en tres módulos:

Generador de código intermedio

Este es el primer paso, su misión consiste en facilitar la traducción del lenguaje fuente al lenguaje objeto traduciendo el primero a un lenguaje intermedio. Utiliza el árbol anotado de la fase de análisis.

Su otra misión es muy útil para reutilizar módulos ya que podemos traducir muchos lenguajes al mismo lenguaje intermedio.

En el ejemplo:

C.I → temp1 := id2*id3

temp2:= 1

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

- - -

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

pasos anteriores. Por ello es necesario el optimizador, para convertir el CI en un lenguaje eficiente:

O.C. → temp1 := id2*id3

id1 := temp1 - 1

Generador de código

El último paso del proceso de síntesis es muy similar al primero, el generador de código se encarga de traducir el CI optimizado resultado del proceso anterior, que es más fácil que el lenguaje inicial, en código ensamblador de muy bajo nivel.

En el ejemplo:

G.C. → move id2, r1

mul id3, r1

sub #1, r1

move r1, id1

Lenguajes y gramáticas

Repaso de la teoría de lenguajes

- **Alfabeto:** Conjunto finito de símbolos. (a-z, A-Z)
 - **String o cadena:** Cualquier secuencia de símbolos de un alfabeto. hola
- **Longitud:** Número de caracteres de una cadena. hola → 4
- **Cadena vacía:** Cadena con longitud 0, denominada λ .
- **Lenguaje:** Conjunto de cadenas válidas formadas por un determinado alfabeto.
 - **Válido:** Cumple unas determinadas reglas.
- **Gramática:** Especificación precisa y formal de todas las cadenas válidas para un determinado alfabeto. Definen y especifican un lenguaje. Se denomina como una 4-tupla (N, T, P, S) donde:

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

$$A \rightarrow 0$$

$$A \rightarrow 0A \rightarrow 00$$

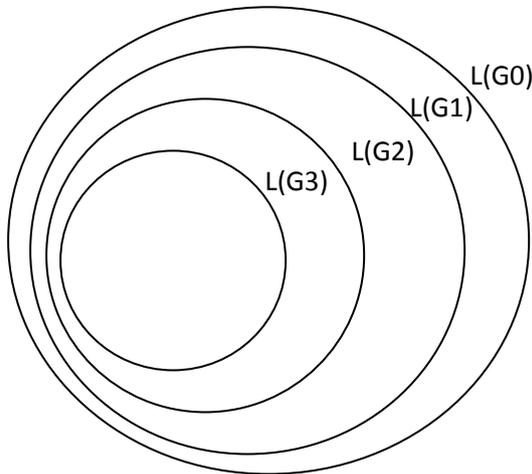
$$A \rightarrow 1A \rightarrow 10A \rightarrow 100$$

Entonces:

$$L(G1) = \{ C / A \rightarrow^* C \} = \{0,1\}^*0$$

Jerarquía de Chomsky

- **Tipo 0 (sin restricciones)** = Las reglas de producción pueden ser de cualquier tipo. $\alpha \rightarrow \beta$ $\alpha, \beta \in \{N \cup T\}^*$
 - **Tipo 1 (dependientes del contexto)** = $\alpha A \beta \rightarrow \alpha \gamma \beta$ $A \in N$ $\gamma \in \{N \cup T\}^*$
 - **Tipo 2 (Independientes del contexto)** = $A \rightarrow \gamma$
 - **Tipo 3 (gramáticas regulares)** = $A \rightarrow aB$; $A \rightarrow a$ $a \in T$ $B \in N$
- } Nos interesan en esta asignatura.



Las gramáticas de tipo 2 las usaremos en el análisis sintáctico.

Para el análisis léxico utilizaremos gramáticas de tipo 3 (regulares)

R. por la izquierda

$$B \rightarrow B\alpha$$

Ejemplo:

$$B \rightarrow B1$$

$$B \rightarrow B1 \rightarrow B11$$

Existen analizadores sintácticos incompatibles con esta recursividad por eso debemos crear una gramática equivalente sin recursividad por la izq.

Eliminación Recursividad por la izquierda

$$\begin{array}{l} A \rightarrow A\alpha \\ A \rightarrow \beta \end{array} \rightarrow A \rightarrow B$$

$$\begin{array}{l} A \rightarrow \beta \\ A' \rightarrow \alpha \end{array} \rightarrow \begin{array}{l} A \rightarrow \beta \\ A \rightarrow \beta A' \end{array} \leftarrow \text{Tras } \beta \text{ hay tantos } \alpha \text{ como se quiera}$$

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

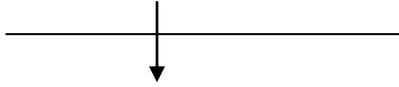
ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Factorizar por la izquierda

$A \rightarrow \alpha\beta$ $A \rightarrow 1B$ Existen

$A \rightarrow \alpha\Upsilon$ $A \rightarrow 1C$



$A \rightarrow \alpha A'$

$A' \rightarrow \beta$

$A' \rightarrow \Upsilon$

Gramática ambigua

Una gramática es ambigua si existen dos formas diferentes de generar una misma sentencia, una misma cadena. Los procesadores del lenguaje no son compatibles con las G. ambiguas.

Ejemplo:

1. $S \rightarrow \text{if } E \text{ then } S$
2. $S \rightarrow \text{if } E \text{ then } S \text{ else } S$
3. $S \rightarrow P$

y en el lenguaje aparece:

if e1 then if e2 then P1 else P2

de forma:

$S \rightarrow^1 \text{if } e1 \text{ then } S \rightarrow^2 \text{if } e1 \text{ then if } e2 \text{ then } P1 \text{ else } S \rightarrow^3 \text{if } e1 \text{ then if } e2 \text{ then } P1 \text{ else } P2$

pero:

$S \rightarrow^2 \text{if } e1 \text{ then } S \text{ else } S \rightarrow^1 \text{if } e1 \text{ then if } e2 \text{ then } S \text{ else } S \rightarrow^3 \text{if } e1 \text{ then if } e2 \text{ then } P1 \text{ else } S \rightarrow^3 \text{if } e1 \text{ then if } e2 \text{ then } P1 \text{ else } P2$

Hemos llegado a la misma cadena mediante dos caminos generadores. Esta es una gramática ambigua. Ningún analizador sintáctico permite una gramática ambigua ya que esta generaría distintos árboles y, por tanto, distintos códigos objeto.

Ejercicio

$E \rightarrow E + T$

$E \rightarrow T$

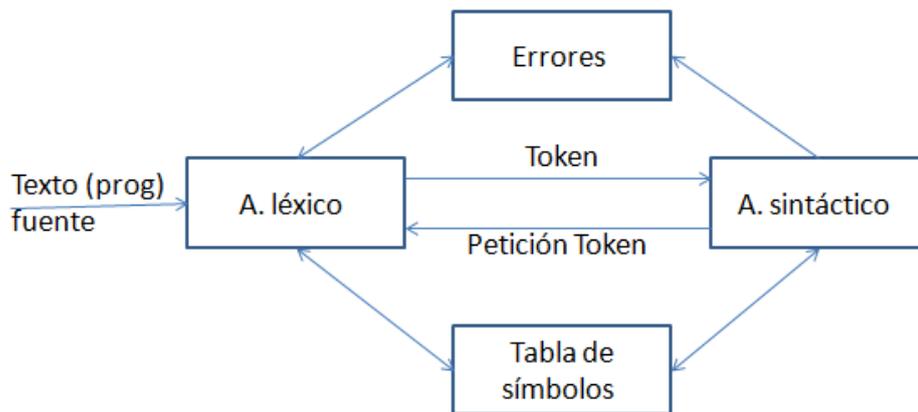
$T \rightarrow T * F$

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Análisis léxico



El analizador léxico simplifica el trabajo al Analizador sintáctico agrupando todas aquellas cadenas que tienen la misma estructura sintáctica. Por ejemplo, si recibe la cadena *límite* el A. léxico debe saber que esto es un **identificador** a nivel sintáctico y así con cada uno de los tokens.

Funciones del Analizador léxico

- Manejar el fichero del texto fuente (TF).
- Leer los caracteres del TF.
- Enviar al Analizador sintáctico un token.
- Eliminar del TF todo lo que sea superfluo. Por ejemplo: comentarios, espacios en blanco, saltos de línea...
- Detectar errores léxicos. Es decir, secuencias de caracteres que no corresponden a ningún token. Por ejemplo:
 - 10,00,0 → Número mal formado
 - 7a8 → Error léxico
 - año, alá → Caracteres incorrectos en la mayoría de compiladores.
- Relacionar los errores con su posición en el TF.
- Pre procesar el TF (macros)
- Interactuar con la tabla de símbolos.

Definiciones

Token (componente léxico)

Cada uno de los elementos del lenguaje que estamos identificando que tienen sentido o significado por sí mismos. Es un conjunto mínimo de caracteres. Ejemplo:

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Todo *token* tiene algunos componentes importantes:

Lexema

Es el conjunto de caracteres que forman un *token*. Ejemplo:

límite (6 caracteres) en un identificador.

Patrón

La regla que describe como un determinado lexema se asocia a un determinado *token*.

Ejemplo:

“Secuencia alfanumérica (sin espacios ni signos de puntuación) que empieza por una letra” en un identificador.

Tipos de token

Existen algunos tipos básicos que aparecen de forma habitual:

- Identificador
- Número
 - Número real
 - Número complejo
 - Número entero
- Operadores
 - Asig. :=
 - Relac. <, >, =, <=, ...
 - Aritm. +, -, /, *, **, ...
- Puntuación → ; , {, }, (,), ... (Aunque hay que saber que cada uno de estos símbolos serían *tokens* distintos ya que se diferencian a nivel sintáctico.)
- Palabras clave → *If, Do, While, ...* Como antes, aunque las hemos agrupado juntos, lo normal es que cada palabra clave forme un *token*.

Aunque existen muchos más.

Un Ejemplo

$a + b * c$ $\xrightarrow[\text{léxico}]{\text{Analizador}}$ $id1/op.aritm1/id2/op.aritm2/id3$ $\xleftarrow[\text{léxico}]{\text{Analizador}}$ $j - i / d$

Como puede verse, ambas sentencias producen el mismo listado de *tokens* a pesar de ser muy diferentes. Si se enviaran al Analizador sintáctico se perdería la información de esa sentencia.

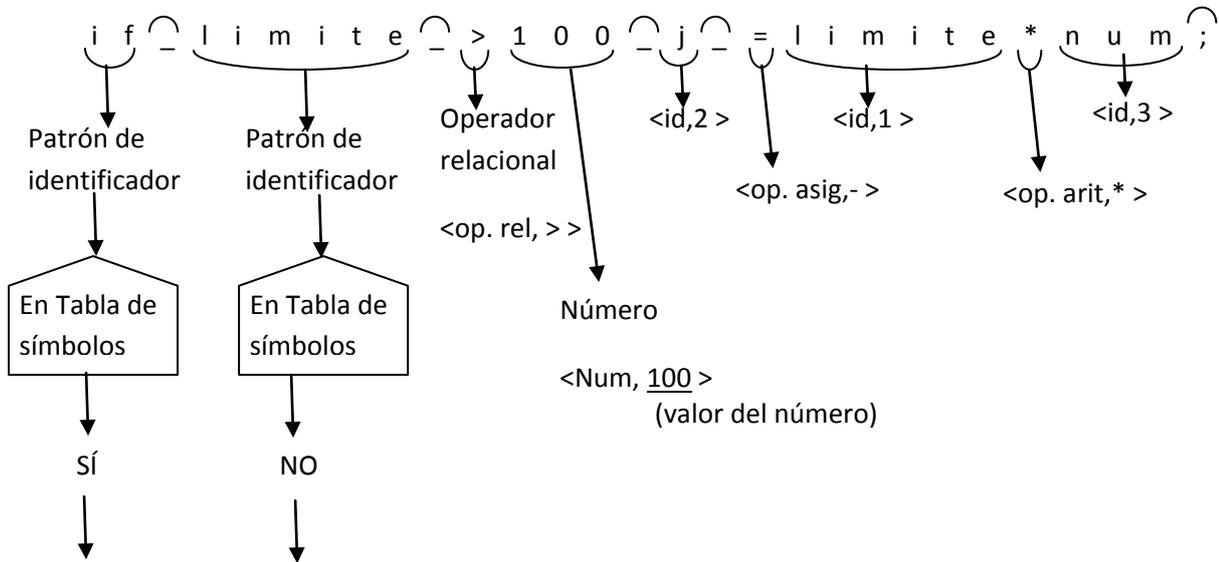
Por eso debemos enviar más información.

The logo for Cartagena99 features the text 'Cartagena99' in a stylized, blue, serif font. The '99' is significantly larger and more prominent than the rest of the text. The logo is set against a light blue background with a subtle gradient and a soft shadow effect.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Ejemplo de funcionamiento



Palabra clave
 <if, ->
 Con '-' se precisa que no es necesaria más información

Se coloca en la tabla de símbolos como identificador1
 <id, 1>
 Posición en TS

Tabla de símbolos

Pos	Lexema	Tipo	...
1	Límite		
2	J		
3	Num		
...

Diseño léxico

Todo analizador léxico dependerá del lenguaje sobre el que se va a diseñar. Cualquier diseño sigue un número determinado de pasos:

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70



5. Tratar errores (incluyendo mensajes de error).
6. Implementar el AFD.

Ejemplo

Lenguaje → Secuencia de identificadores separada por blancos. El patrón de los identificadores es el descrito anteriormente.

Por ejemplo:

límite_j_ind13

hola_estoy____aqui

Paso 1

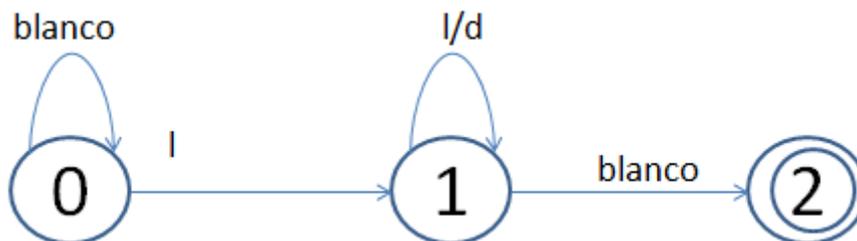
Tokens: <id, pos_TS>

Paso 2

<N, T, P, S> → P: $\begin{cases} S \rightarrow lA \text{ (l es cualquier letra l = a-z, A-Z)} \\ A \rightarrow lA / dA \text{ (d es cualquier dígito d=0-9)} \\ A \rightarrow \lambda \end{cases}$

Entonces: <{S, A}, {l, d}, P, {S}>

Paso 3



Paso 4

Acciones semánticas

L : leer siguiente carácter ← Trivial, pero necesario especificar.

C': lexema es la l

C : lexema es el lexema o la concatenación leída l/d

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

L, C

L, Buscar

Paso 5

Llegados a este punto definiremos que cualquier cosa que no contemple el autómata es un error, sin necesidad de especificarlo de forma explícita. Sin embargo, habría que definir los

Ejercicio

Hacer un diseño igual que este para el lenguaje:

Secuencias de n^{os} reales separados por blancos sabiendo que:

n^{o} real = entero[, decimal]

Ejemplo de analizador léxico 1

Paso 1: Tokens

- := \rightarrow <op.asig, - >
- +, * \rightarrow <op. arit, +>, <op. arit, *>
- > \rightarrow <op. rel, >>
- identificador \rightarrow < id, pos_TS>
- n^{os} enteros \rightarrow <nume, valor>

Paso 2: Gramática regular

<N, T, P, S>

P: $\left\{ \begin{array}{l} S \rightarrow :A/+/*/>/IB/dC/\text{del } S \rightarrow \text{Del } S \text{ no forma parte del proceso de la gramática, ayuda al autómata.} \\ A \rightarrow = \\ B \rightarrow IB/dB/\lambda \\ C \rightarrow dC/\lambda \end{array} \right.$

l = letra= a-z, A-Z

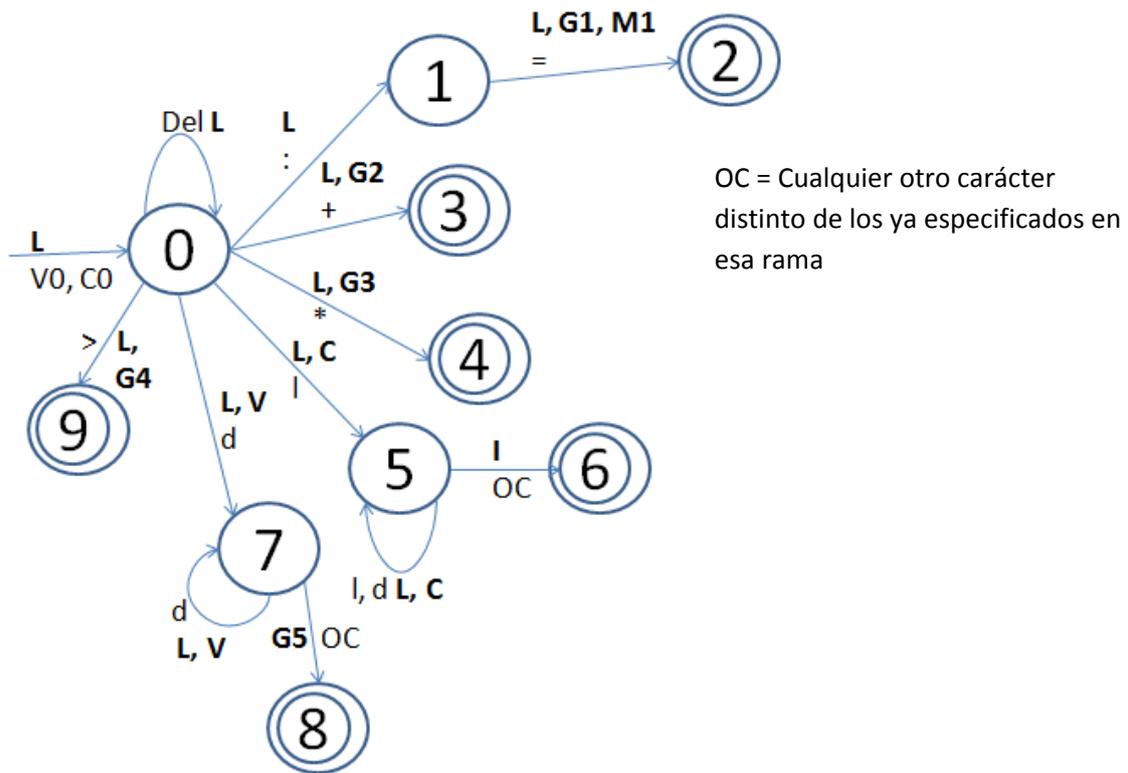
d= dígito = 0-9

The logo for Cartagena99 features the text 'Cartagena99' in a stylized, blue, serif font. The '99' is significantly larger and more prominent than the 'Cartagena' part. The text is set against a light blue background with a subtle gradient and a soft shadow effect.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Paso 3: Autómata finito



Paso 4: Acciones semánticas

- L = leer siguiente carácter.
- C0 = lexema = " " (iniciamos variable)
- C = concat (lexema, l/d)
- I = posición := buscar (lexema, TS)
If posición = null then posición := insertar (lexema, TS)
Generar token (id, posición)
- V0 = valor = 0 (iniciamos variable)
- V = valor * 10 + d = valor ('d'=valor del dígito)
- G1 = Generar token (op.asig, -)
- G2 = Generar token (op. arit, +)
- G3 = Generar token (op. arit, *)
- G4 = Generar token (op. rel, >)
- G5 = Generar token (num_l, valor)

Paso 5: Errores

Los mensajes de error se crean y envían desde el módulo de control de errores. Un analizador

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

Hay dos formas básicas de manejar un error, descubrirlo y terminar (que es la forma fácil) o volver al principio para subsanar el error.

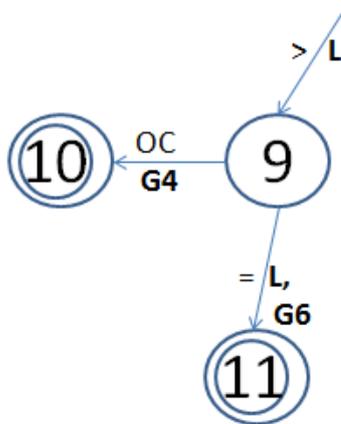
Ejercicio

¿Qué cambios habría que realizar en los puntos 3 y 4 para tratar palabras reservadas?

Añadidos al ejemplo

Un cambio que podríamos realizar es añadir un op. relacional nuevo: ">=". En el autómata modificamos la rama del op. relacional (estado 9), de la siguiente forma:

Y añadir la acción semántica **G6N**=Generar token(op.rel., >=)



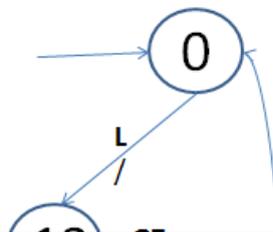
Ahora vamos a seguir añadiendo más características típicas de un lenguaje de programación a este modelo. Por ejemplo:

- Comentarios: `/* esto es un comentario */`
- Cadenas de caracteres: `"esto es // una /"cadena=` → `esto es /una "cadena`

Con la doble barra añadimos una barra a la cadena y una barra con comillas sirve para añadir comillas a la cadena.

Comentarios

Para los comentarios debemos añadir al lenguaje `'/`



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

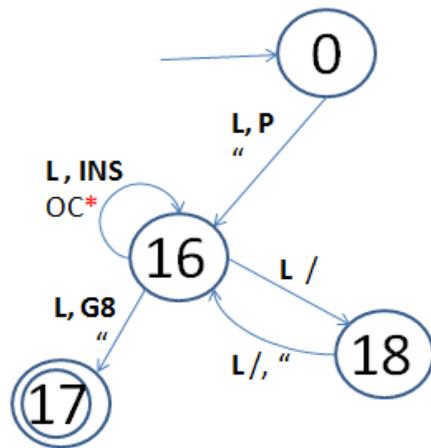
Cartagena99



que estamos, añadir el operador división.

Cadena

La cadena es algo diferente porque sí genera un token: token cadena <cadena, puntero a memoria> Se almacena en memoria debido a su tamaño, demasiado grande para la TS. Debemos añadir al lenguaje |"|" (comillas dobles)



Se añaden las Acciones semánticas:

G8 = Generar token (cadena, punt. mem.)

P = Reserva memoria.

INS = Concatena el contenido de la cadena.

*Salvo " y /

Ejemplo de analizador léxico 2:

Paso 1

Analizador de números reales.

$\text{dig}^+[\cdot\text{dig}^+][\text{e}[\text{signa}]\text{dig}^+]$

- <entero, valor>
- <real, valor>

Paso 2

Gramática:

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

- - -

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

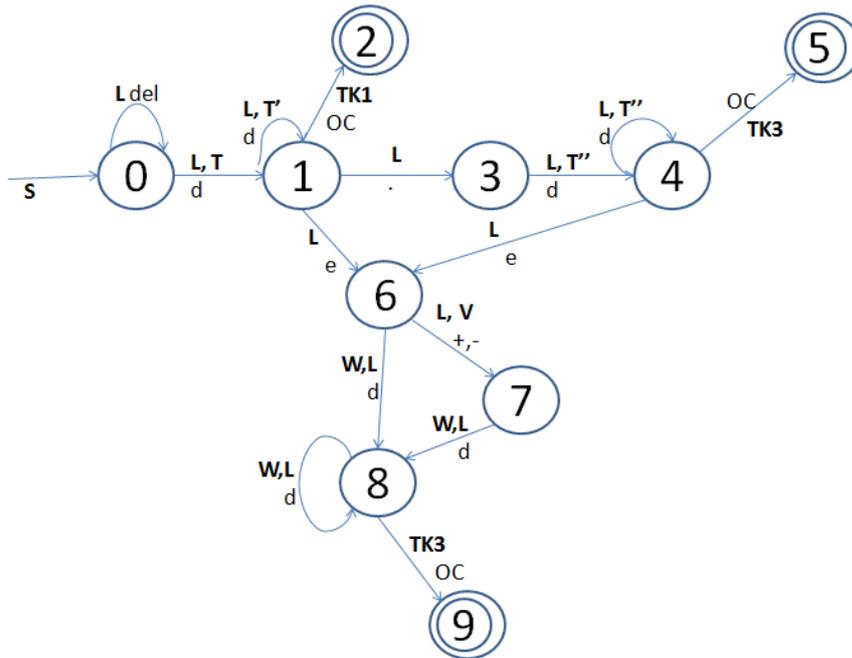
$D' \rightarrow dD'/\lambda/eE$ (bucle para crear más decimales/fin de la parte decimal sin exponente/exponente tras la parte decimal)

$E \rightarrow +E'/-E'/dE''$ (exponente positivo/exponente negativo/sin signo)

$E' \rightarrow dE''$ (aseguras un dígito tras el signo)

$E'' \rightarrow dE''/\lambda$ (bucle para crear más exponente/fin del exponente)

Paso 3



Paso 4

Acc. semánticas

L = Leer siguiente carácter (Iría en todas las transiciones excepto en OC)

S : Iniciamos variables.

- num = 0
- exp = 0
- ndec = 0
- sig = +1
- Tipo = null

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

→12,57 num=1257
 ndec=2

TK2 = valor = $num / 10^{ndec}$
Generar token (real, valor)

TK3 = valor = $num * 10^{(exp + sig) - ndec}$
Generar token (real, valor)

V = if "-" then sig = -1

W = exp = $exp * 10^d$

Ejemplo de Paso 1

Tenemos la siguiente gramática:

$S \rightarrow \text{Switch (E) do \{C\} / break / id := E ; / } \lambda$	}	Reglas sintácticas
$L \rightarrow \text{case N:SL / default : S / } \lambda$		
$E \rightarrow \text{id N -E / (E)}$		
$N \rightarrow \text{cte / -cte}$	}	Reglas léxicas
$\text{id} \rightarrow \text{id l / id d / l}$		
$\text{cte} \rightarrow \text{cte d / d}$		

Haya los tokens necesarios:

- | | | | |
|-----------------------|----------|----------|---------------|
| • <id, pos_TS> | • <{, -> | • <}, -> | • <:=, -> |
| • <cte, valor> | • <}, -> | • <:, -> | • <- -> |
| • <pal_clave, código> | • <{, -> | • <:, -> | • signo menos |

Ejercicio

Dado el siguiente texto en xml:

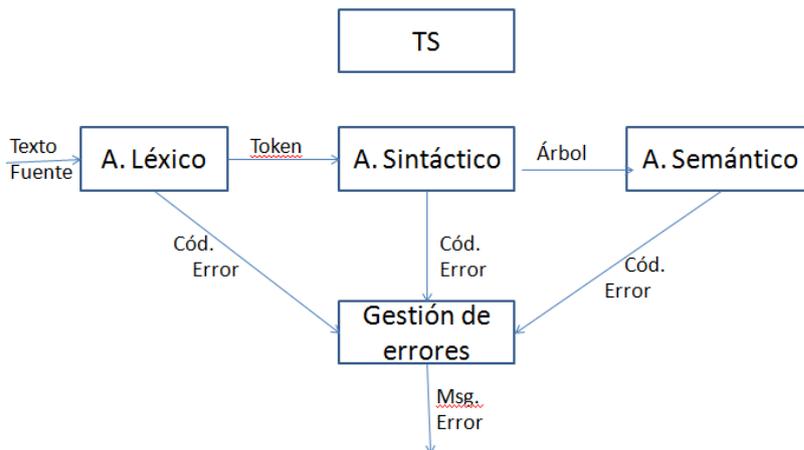
```
<tfc>
  <autor> Luis Gomez </autor>
  <fecha_nac> 18/05/85 </fecha_nac>
  <título> Diseño de An. Léxico </título>
  <autor> Maria Lopez </autor>
```

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

Errores



Cómo tratar los errores

- Obviarlos, que no envíe ningún mensaje. → No se debe utilizar.
- Señalar el error y parar. Es la forma válida más sencilla, pero ineficiente ya que si hay varios errores tendremos que reiniciar el procesador tras solucionar cada uno.
- Señalar el error y continuar buscando más errores (sin generar código). Es más eficiente que el anterior pero tiene el problema de que no siempre un procesador se recupera de un error y podría arrastrarlo y que se generen más errores.
- Señalar el error y tratar de corregirlo. La mejor forma, pero la más compleja.

Errores típicos

Análisis léxico

- Caracteres que no pertenecen a los terminales. *Año*
- Identificadores mal formados. *3ab*
- Constantes mal escritas. *3.0.4*
- Constantes fuera de límite. *12975432*

Análisis sintáctico

- Símbolos no permitidos en esa posición. *If... then 3Begin...*
- Ausencia de delimitadores. *i=3__f=4;*
- Palabras reservadas mal formadas. *if...Dhen...*

Análisis semántico

- Incompatibilidades de tipo en expresiones/asignaciones. *entero = entero*3,0*
- Inconsistencia de tipo entre parámetros formales y actuales.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

- Localizar los errores adecuadamente.
- Completos y legibles.
- Amables y respetuosos.

Tabla de símbolos

La tabla de símbolos (TS) es el lugar en el que se guarda, de forma ordenada, todos los identificadores que existen en el texto fuente. La TS es una estructura de datos que está presente mientras se ejecuta el compilador, entonces, obviamente, no existe cuando se ejecuta el programa. Esta es una cuestión crítica.

Tipos de información

- Lexema. La secuencia de caracteres que representa al identificador en el texto fuente.
- Tipo del identificador.
- Índice, aunque es más una característica de la propia TS. Suele almacenarse en una estructura dinámica.
- Alcance.
- Dimensiones.
- Argumentos (procedimiento o función).
- Tipo de valor devuelto (función).
- Dirección de memoria estática, donde se almacena el valor de la variable cuando se ejecute el programa. Enlace. El propio valor nunca se guarda en la tabla ya que es necesario en ejecución, cuando la TS ya no existe.

La TS no es una tabla estática ya que la cantidad de memoria que se necesita para guardar cada identificador es variable, depende del número de identificadores y la cantidad de información que tengan. La TS es, por tanto, una tabla dinámica.

Organización de la TS

Imaginemos los siguientes identificadores: A, Aux, I, Cont, X

Podemos organizarlos en diferentes TS:

TS Lineal

Se introducen de forma lineal en la tabla.

Iden.	Lexema	Tipo	...

Para meter el identificador el corte es trivial, pero no así para buscar dentro de la TS ya que, en el caso

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

TS ordenada

Iden.	Lexema	...
1	A	...
2	Aux	...
3	I	...
4		

Cont > I

Iden.	Lexema	...
1	A	...
2	Aux	...
3	Cont	...
4	I	...

A > Aux > I

TS hash

En este caso la tabla se ordena según una función hash que le asigna un índice.

Función → Número de caracteres (función hash un poco inútil).

Iden.	Lexema	...
1	A	...
2		
3	Aux	...

A → 1
Aux → 3
I → 1

Ya está siendo usada (la función hash no es perfecta)

Para evitar esto existen varias soluciones:

1. El identificador se coloca en la siguiente posición vacía ← No es la mejor.
2. Se crea una tabla de desbordamiento a la que las posiciones con varios identificadores se dirigen cuando se busca o se inserta un identificador de esa posición.

Iden.	Lexema	...	Desbordamiento
1	A	...	
2			
3	Aux	...	

Con = 3

Lexema	Tipo	...	Desbordamiento
I	
X	...		
Con	

Solo se enlazan los identificadores con igual índice.

La evolución natural de esta solución es que la TS sea una lista de índices que enlacen con el primer identificador de ese índice en la tabla de desbordamiento.

Iden.	Des.
1	

Lexema	...	Desbordamiento
A	...	

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Manejo de errores con la TS

```

1 Programa básico
2 Var a:integer
3 Procedure suma (a:integer, b:integer)
4     Begin
5         a:=a+b
6     End
7 End
8 Begin
9     Read (a, b)
10    Print (suma (a,b))
11 End
    
```

Problema 1 En este caso existen dos variables 'a', una dentro del *Procedure* y otra global. El compilador considera que esta variable es la más cercana, en este caso la que está dentro del proceso. Debe solucionarse también con la TS.

Problema 2 Identificador no declarado. En este caso se debe saber esto gracias a la TS y conociendo el alcance de dicho identificador.

Solución del 1^{er} problema

Doble variable en el texto. Para solucionarlo la opción más común es crear **otra** TS solo para el alcance más pequeño de una de las variables, en este caso, el procedimiento suma.

TS Global		TS Suma	
a	Integer	a	Integer
		b	Integer

Durante el proceso de compilación se utilizará la TS de la zona en la que estemos cada vez. Hasta la línea 3 se utiliza la TS actual es TS Global. Al encontrar la palabra *procedure* (por ejemplo, hay que saber que palabras clave crean una TS) crearemos una nueva TS con el nombre de ese proceso y la TS actual pasa a ser dicha TS.

Esta es la forma de conocer el alcance de un identificador → Solución 2^o problema.

Cartagena99

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Ejercicio resuelto

Dado este programa

```
int a,b; /*globales */

void proc1 (int *x)
{
    int b;
    b = 3;
    *x = b * 2 + a;
}

boolean fun (int x)
{
    int a;
    a = 2;
    return (u % a) ==
b;
}

void proc2 (int d)
{
    int u;
    boolean v;
    u = a + d;
    v = fun (u);
}

void main ()
{
    a = 1;
    proc1(&a);
    proc2(&a);
    fun (b);
}
```

Normalmente ejercicios de este tipo piden determinar cómo avanza la TS.

Se definen durante la evolución:

- Tabla actual (TA) → Que indica que TS se desarrolla en ese momento.
- Definición/uso (D/U) → Que indica en que zona estamos, o en definición de variables o en uso de ellas.

Primero se crea la TS global.

Id.	Lexema	tipo	Valor Devuelto
1	a	Int.	-
2	b	Int.	-
3	proc1	Función	Void
4	fin	Función	Boolean

TS Global al final del ejercicio

Comienza el proceso, se inicia el analizador sintáctico.

TA → TS Global

D/U → D

Se lee primero <palClave, int> que es una palabra clave. Después lee 'a' <id, 1> busca en la tabla de símbolos y al no estar lo inserta (por estar en zona de definición de variables). Lee <, -> y después <id, 2> que se inserta. Continúa leyendo <; , ->, el comentario se salta, y después <palClave, void>. A continuación de 'void' sabemos que comenzará una zona nueva de definición, entonces se crea la tabla de proceso. <id, (proc1)> → <id, 3> este token se define en la TS Global porque puede accederse desde cualquier lugar del programa.

Id.	Lexema	tipo	...
-----	--------	------	-----

Ahora es cuando se cambian los flags:

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

zona de definición (D/U → D). Se añade el identificador 'b' y se vuelve a la zona de uso (D/U → U). Entonces se lee la variable 'a', al estar en zona de uso se busca en la TS actual (TS proc1) al no estar se busca en la TS Global, como está se envía el *token* <id, 1(TS Global)>. Cuando se llega al símbolo cierre de llave la TS proc1 se borra, ya no hace falta más y el flas TS pasa a ser TS Global.

Se continuaría de esta forma, creando a continuación la tabla fun para la función fun. Lo único destacable es que al leer el carácter 'u' en zona de uso en la TS fun y no estar en ninguna tabla saltaría un error.

Procesos anidados

Dado el ejemplo:

```
PROGRAM Ejemplo;
  VAR a, b : INTEGER;

PROCEDURE proc1 ( VAR x: INTEGER);
  VAR b : INTEGER;
BEGIN
  b := 3;
  x := b * 2 + 4;
END;

PROCEDURE proc2 (d : INTEGER);
  VAR u : INTEGER;
  v : BOOLEAN;
  FUNCTION fun (x : INTEGER) : BOOLEAN;
  VAR a : INTEGER;
  BEGIN
    a = 2;
    return ( u MOD a) = b;
  END;
BEGIN
  u := a + d;
  v := fun (u);
END;
BEGIN
  a := 1;
  proc1 (a);
  proc2 (a);
  fun (b);
END;
```

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

Paso 1

Tabla bloques

Bloque	Bloque padre	Puntero
Global	-	
Proc1	Global	
Proc2	Global	
Fun	Proc2	

TS Global

Ind.	lexema	tipo	Desplazamiento
1	Ejemplo	Program	-
2	a	Entero	0
3	b	Entero	2
4	Proc1	Procedure	-
5	Proc2	Procedure	-

Al leer el primer *Procedure* se crea un nuevo bloque anidado a TS global. Se pasa al paso 2.

Paso 2

TS Proc1

Ind.	lexema	tipo
1	x	Integer
2	b	Integer

Cuando se llega al primer BEGIN se ha terminado de construir la tabla Proc1 y se pasa a la zona de uso.

Al acabar (END) se elimina la TS Proc1 y también de la tabla de bloques. Y se pasa al paso 3

Paso 3

Cuando se lee el siguiente proceso se crea la tabla correspondiente.

TS Proc2

Iden.	lexema	tipo	Valor dev.
1	d	Int	-
2	u	Int	-
3	v	Boolean	-
4	Fun	Function	Boolean

El identificador Fun crea un bloque nuevo, hijo de Proc2. Se pasa al paso 4.

Paso 4

Iden.	lexema	tipo
1	x	Int
2	a	Int

'a' ya existe en otros bloques, pero al estar en zona de definición es como un lexema nuevo.

Cuando se pasa a zona de uso y se encuentra BEGIN se encuentra el carácter u, se busca en *fun*, al no estar se va al bloque padre de *fun*, *proc2*, donde sí existe, entonces se usa ese. Con

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

```
Struct Fecha
{int día, mes, año;
}
Fecha x; ← Registro de tipo fecha.
```

mimes = x.mes ← Acceso a campos del registro.
son dos tokens

Tabla de símbolos actual

B	lexema	tipo	Tamaño reg.	campos
...	No importa lo que ya haya en la TS	
...		
n	Fecha	Registro	6	
n+1	x	Fecha	-	-

TS Fecha

	lexema	tipo	Despl.
1	Día	Int	0
2	Mes	Int	2
3	Año	Int	4

Al leer los diferentes campos que habrá en un registro. La mejor forma de representarlo es añadir una TS adicional con el identificador fecha que indexe los campos del registro.

Al cerrar el registro fecha la TS Fecha se da por finalizada y la suma de su tamaño (desplazamiento) será el tamaño del registro.

Lo siguiente que sucede es la creación de una fecha. Como siempre se añade a la TS de forma normal.

Entonces, ahora se quiere acceder al registro X, como siempre se busca los *tokens* para enviar.

<id, TS(n+1)>

<punto, -> → En este momento se cambia la TS actual a TS Fecha. Ya que se va a querer acceder a ella.

<id, TSFecha(2)> → Tras esto se vuelve a la TS General.

Con este tipo de organización de la TS se sabe siempre que tabla se está usando y se elimina la confusión, permitiendo doble asignación de variables, por ejemplo.

Array (8) of Fecha y; Se crea un vector de 8 registros Fecha.

Se introduciría en la tabla de símbolos de forma corriente.



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

El analizador sintáctico necesita una gramática más avanzada que el léxico. El léxico utilizaba una gramática tipo 3, el A. Sintáctico utilizará una gramática de tipo 2 (independiente del contexto), donde:

$$A \rightarrow \alpha \quad A \in N \\ \alpha \in (N \cup T)^*$$

Es decir, que la única restricción es que a la izquierda haya un no terminal.

El resultado de trabajo en este módulo es un árbol sintáctico que comprenda si es correcto o no y donde estén los errores. Existen dos formas de construir el árbol:

- Construcción ascendente → En esta asignatura: Analizador Sintáctico Ascendente LR
- Construcción descendente → En esta asignatura: { Predictivo – recursivo.
LL o por tablas.

Veamos un ejemplo:

G : (E, N, T, P)

N = {E, T, F}

T = {+, *, (,), id}

P = E → E+T (1)

E → T (2)

T → T*F (3)

T → F (4)

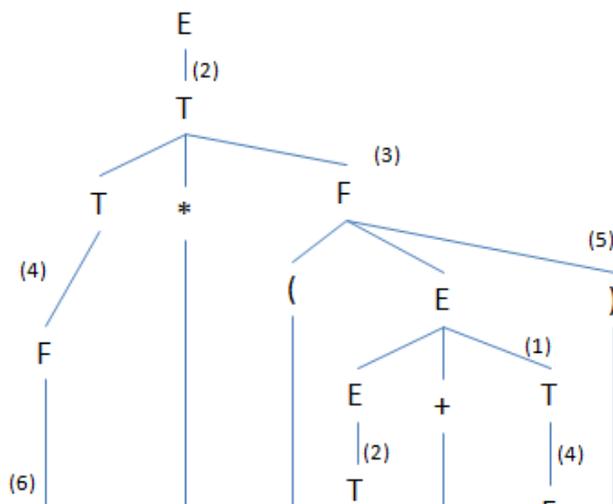
F → (E) (5)

F → id (6)

Dada la cadena:

id * (id + id)

Método descendente



Siempre se expande el no terminal más a la izquierda, por convenio.

Esta es la forma gráfica de representar el árbol, existe también una manera de hacerlo más textual, solo con el número de las producciones efectuadas:

{2 3 4 6 5 1 2 4 6 4 6}

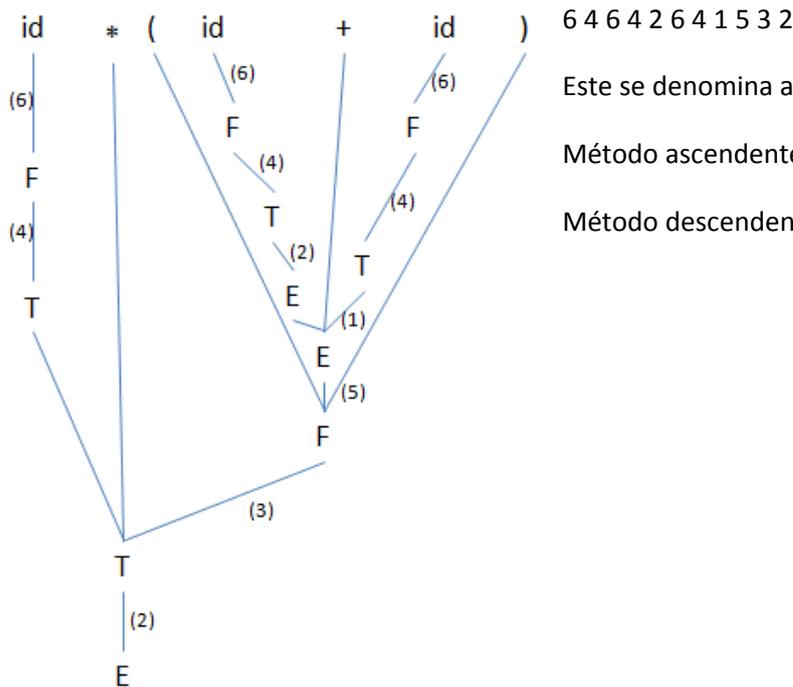
Este método se denomina Parse o análisis a izquierdas debido a la forma de expandir.

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

Método descendente



Este se denomina análisis a derechas.

Método ascendente → arriba a derecha.

Método descendente → arriba a izquierda.

Gramática ambigua

1. $E \rightarrow E + E$ [id + id * id]
2. $E \rightarrow E * E$
3. $E \rightarrow id$

Análisis ascendente

$\underline{id} + id * id \xrightarrow{3} E + \underline{id} * id \xrightarrow{3} \boxed{E + E * id} \rightarrow E + \underline{E * E} \xrightarrow{1} \underline{E + E} \xrightarrow{1} E$
 pivote

Sería equivalente:

$\underline{id} + id * id \xrightarrow{3} E + \underline{id} * id \xrightarrow{2} \boxed{E + E * id} \xrightarrow{1} E * \underline{id} \xrightarrow{3} \underline{E * E} \xrightarrow{2} E$

Una gramática ambigua es una gramática para la cual existe más de un árbol que la genere. No nos interesan las gramáticas ambiguas ya que queremos que el compilador siempre funcione de la misma manera.

Una gramática no ambigua de la anterior es:

$E \rightarrow T / E + T$

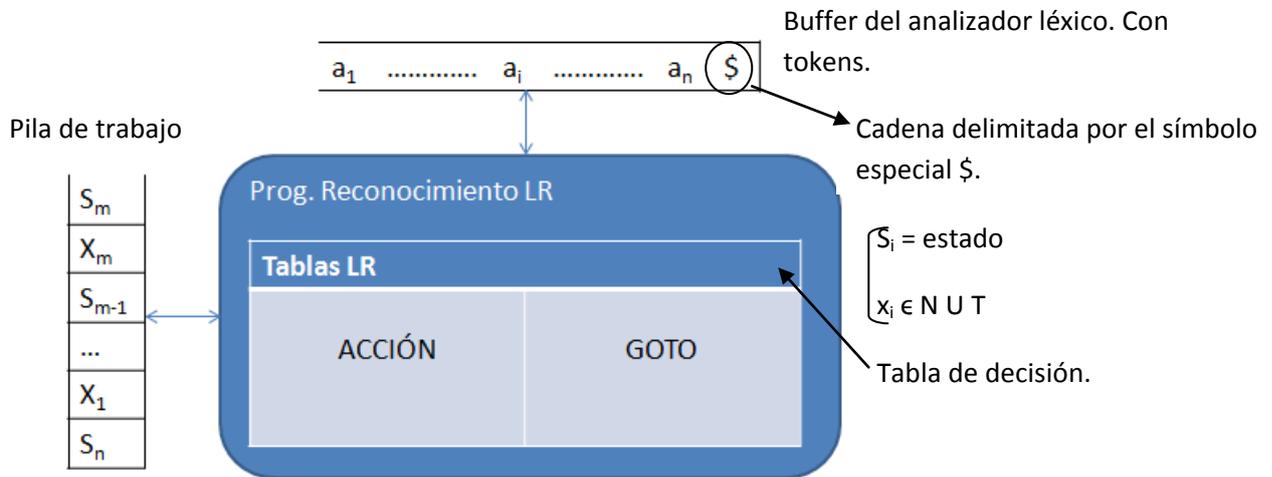
CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70

Cartagena99

Análisis ascendente por reducción/desplazamiento: LR

Existen varios tipos de LR (SLR, LR canónico, LALR), nosotros vamos a utilizar el SLR. Este método es suficientemente general como para reconocer cualquier construcción que existe hoy en día en los lenguajes de programación. Detecta los errores lo antes posible.



El inconveniente que tiene es que es el método más engorroso de construir.

Tabla de decisión

Acción [Estado x token]

$$\text{Acción } (S_j, a_i) = \begin{cases} \text{Reducir } A \rightarrow \alpha \\ \text{Desplazar y transitar al estado } S_k \\ \text{Error (Código error).} \\ \text{Aceptar.} \end{cases}$$

Reducir

$$(S_0, x_1, \dots, x_m, S_m; a_1, \dots, a_n \$) \rightarrow (S_0, x_1, \dots, x_{m-r}, S_{m-r}, A, S; a_1, \dots, a_n \$)$$

$$x_{m-r+1}, \dots, x_m = \alpha \quad A \rightarrow \alpha B$$

$$S \leftarrow \text{GOTO } (S_{m-r}, A)$$

Desplazar

Significa pedirle un *token* al léxico y meterlo en la pila.

$$(S_0, x_1, \dots, x_m, S_m; a_1, \dots, a_n \$)$$

$(S_0, x_1, \dots, x_m, S_m; a_1, \dots, a_n \$) \rightarrow (S_0, x_1, \dots, x_m, S_m, a_{n+1}, S_{m+1}; a_1, \dots, a_n \$)$ se introduce nuevo que nunca puede haber un

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

Tabla de decisión

estado	ACCIÓN						GOTO		
	id	+	*	()	\$	E	T	F
S ₀	D5				D4		1	2	3
S ₁		D6				ACEPTAR			
S ₂		R2	D7		R2	R2			
S ₃		R4	R4		R4	R4			
S ₄	D5				D4		8	2	3
S ₅		R6	R6		R6	R6			
S ₆	D5				D4			9	3
S ₇	D5				D4				10
S ₈		D6				D11			
S ₉		R1	D7		R1	R1			
S ₁₀		R3	R3		R3	R3			
S ₁₁		R5	R5		R5	R5			

D = desplazar R = reducir

PILA	Cadena	ACCIÓN
S ₀	<u>id</u> * id \$	D5
S ₀ idS ₅	<u>*</u> id \$	R6 (F → id)
S ₀ FS ₃	<u>*</u> id \$	R4 (T → F)
S ₀ TS ₂	<u>*</u> id \$	D7
S ₀ TS ₂ *S ₇	<u>id</u> \$	D5
S ₀ TS ₂ *S ₇ idS ₅	\$	R6 (F → id)
S ₀ TS ₂ *S ₇ idS ₅	\$	R3 (T → T*F)
S ₀ TS ₂	\$	R2 (E → T)
S ₀ ES ₁	\$	ACEPTAR

A la hora de tomar decisiones se tiene en cuenta el estado y el *token* actual. Mirando la tabla de decisión: S₀ con id indica D5, desplazar al estado 5. Entonces se añade el *token* a la pila, se elimina de la cadena y como siempre debe haber un estado en la cima de la pila añadimos el estado 5.

Ahora tenemos estado 5 y * como *token*, eso nos da R6 (reducir por la regla 6) eso significa reducir por F → id, en la pila se eliminan tantos símbolos como el doble de

elementos a la derecha de la regla, en este caso hay un símbolos (F → id), por tanto en la pila se eliminan dos símbolos (id, S₅) uno de los símbolos debe coincidir con el que está a la derecha de la regla, en este caso es correcto. Entonces en la pila se coloca el símbolo a la izquierda de la regla (S₀F) y para conocer qué estado hay que poner a continuación se va a la tabla GOTO con el estado en la pila (S₀) y el símbolo (F) entonces nos lleva al estado 3. Así vamos utilizando el algoritmo LR y poco a poco se llega a la situación de la pila final.

Construcción de tablas de decisión LR

Prefijo viable

Son todos los prefijos de una forma sentencial que no van más allá del pivote.

Dada la cadena:

El pivote es la subcadena que vamos a tomar para reducir y que forma parte del camino correcto.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

La base de la construcción de las tablas de decisión LR es encontrar todos los prefijos viables de cualquier subcadena del lenguaje. Para ello utilizaremos un autómata reconocedor de prefijos viables.

Ítem

También llamado elemento.

Es cualquier regla de la gramática con un punto a la derecha de la regla.

$E \rightarrow E \cdot + T$
 $E \rightarrow \cdot T$
 $F \rightarrow id \cdot$

Si yo quiero aplicar la regla que me indica el ítem necesito encontrar todos los elementos a la derecha del punto, es decir:

$F \rightarrow \cdot id$ / El A. léxico nos pasa un identificador $\rightarrow F \rightarrow id \cdot$
 Para poder aplicar esta regla necesito un identificador. Entonces ya puedo aplicar la regla.

Dos funciones para el manejo de ítems

Función cierre

Dado una G , I = conjunto de ítems válido.

Cierre (I) : $I \subseteq \text{Cierre}(I)$

$\text{Si } A \rightarrow \alpha \cdot B \beta \in \text{Cierre}(I) \Rightarrow \forall B \rightarrow \gamma \in G, B \rightarrow \cdot \gamma \in \text{Cierre}(I)$
 punto+no terminal

Ejemplo

Siguiendo la gramática habitual:

$I = \{E \rightarrow \cdot E + T\}$

Cierre (I) = $\{E \rightarrow \cdot E + T, E \rightarrow E \cdot + T, E \rightarrow \cdot T, T \rightarrow \cdot T * F, T \rightarrow \cdot F, F \rightarrow \cdot (E), F \rightarrow \cdot id\}$

No se incluyen los repetidos.

El cierre de un estado son todas las posibilidades del lenguaje que se pueden aplicar desde una determinada regla.

Función Goto

Goto (I, x) $x \in N \cup T$



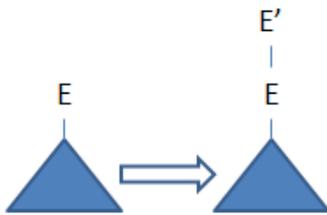
CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70

Gramática aumentada

$G(N, T, P, S) \rightarrow$ Gramática aumentada $\rightarrow G'(N \cup \{S'\}, T, P \cup \{S' \rightarrow S\}, S')$

Es un formalismo que utilizamos al crear un LR. Esta gramática aumentada nos asegura que hemos llegado al final cuando usamos la reducción por la regla $E' \rightarrow E$



Podría haber otras producciones con E, pero solo una con E'.

Autómata reconocedor de prefijos viables

Colección canónica

Conjunto de ítem inicial $I_0 = \text{Cierre}\{S' \rightarrow \cdot S\}$

Colección canónica $CC = \{I_j\} \quad j=1 \rightarrow$ contador

Repetir

$\forall I_i \in CC \quad \forall x \in N \cup T$

$I_j = \text{Goto}(I_i, x)$

Si $I_j \in CC, I_j \neq \emptyset \rightarrow CC = CC \cup \{I_j\}, j = j+1$

Hasta que CC no cambie.

Ejemplo

$I_0 = \text{Cierre}\{E' \rightarrow \cdot E\} = \{E' \rightarrow \cdot E, E \rightarrow \cdot E + T, E \rightarrow \cdot T, T \rightarrow \cdot T * F, T \rightarrow \cdot F, F \rightarrow \cdot (E), F \rightarrow \cdot id\}$

$CC = \{I_0\} \quad j = 1$

$I_0 = \text{Goto}(I_0, E) = \text{Cierre}\{E' \rightarrow E \cdot, E \rightarrow E \cdot + T\} = \{E' \rightarrow E \cdot, E \rightarrow E \cdot + T\} = I_1$

$I_0 = \text{Goto}(I_0, T) = \text{Cierre}\{T \rightarrow T \cdot * F, E \rightarrow T \cdot\} = \{T \rightarrow T \cdot * F, E \rightarrow T \cdot\} = I_2$

$I_0 = \text{Goto}(I_0, F) = \text{Cierre}\{T \rightarrow F \cdot\} = \{T \rightarrow F \cdot\} = I_3$

$I_0 = \text{Goto}(I_0, +) = \text{Cierre}\{\emptyset\} = \{\emptyset\}$

$I_0 = \text{Goto}(I_0, *) = \text{Cierre}\{\emptyset\} = \{\emptyset\}$

$I_0 = \text{Goto}(I_0, '(') = \text{Cierre}\{F \rightarrow (\cdot E)\} = \{F \rightarrow (\cdot E), E \rightarrow \cdot E + T, E \rightarrow \cdot T, T \rightarrow \cdot T * F, T \rightarrow \cdot F, F \rightarrow \cdot (E), F \rightarrow \cdot id\} = I_4$

$I_0 = \text{Goto}(I_0, ')') = \text{Cierre}\{\emptyset\} = \{\emptyset\}$

$I_0 = \text{Goto}(I_0, id) = \text{Cierre}\{F \rightarrow id \cdot\} = \{F \rightarrow id \cdot\} = I_5$

$CC = \{I_0, I_1, I_2, I_3, I_4, I_5, \dots\}$

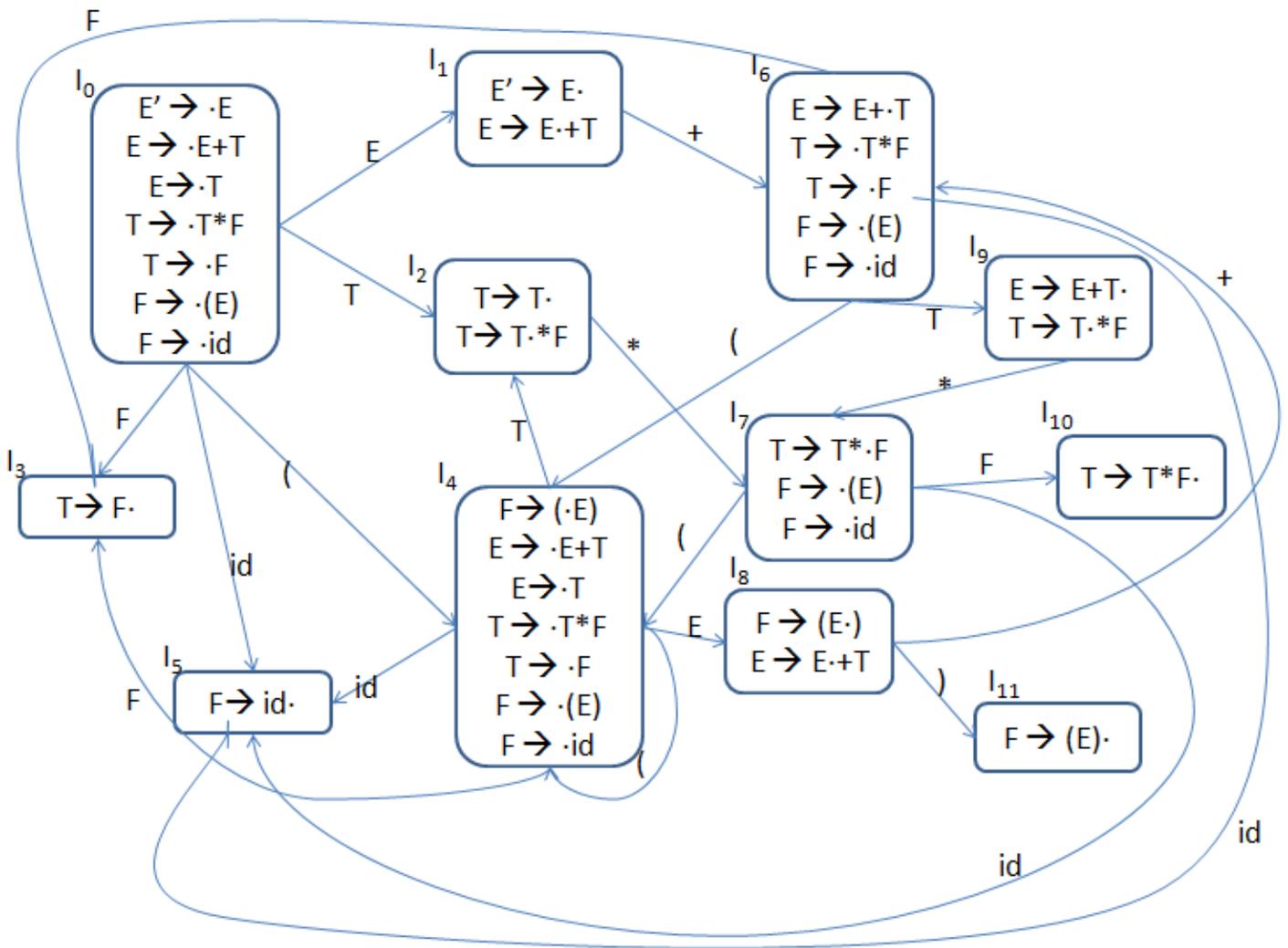
y continuaría, con esto solo terminamos el bucle en el estado I_0

habría que hacer nuevos bucles para cada nuevo estado.

Cartagena99

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70



Ejercicio

Crear un autómata reconocedor de prefijos viables.

- A \rightarrow BC
- B \rightarrow dB / b
- C \rightarrow dB / C / dC

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70

Cartagena99

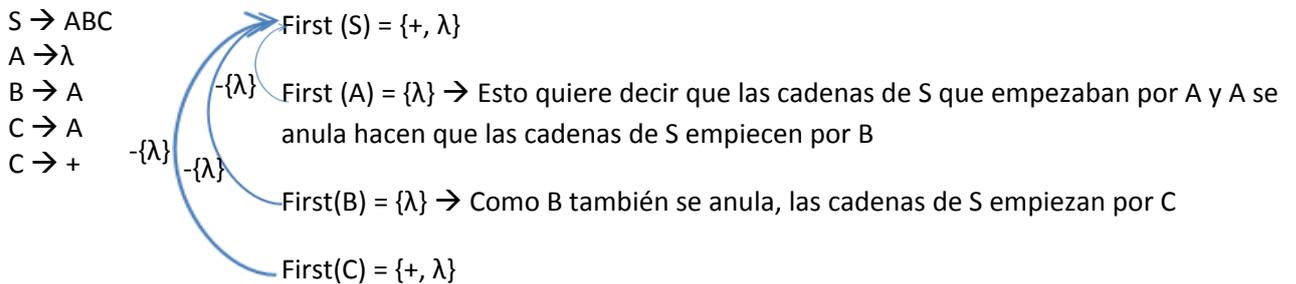
Ejemplo

Dada la gramática habitual:

$E \rightarrow E + T$	$\text{First}(+) = \{+\}$	} El $\text{First}(\alpha)$ donde $\alpha \in T$ es siempre y solo α .
$E \rightarrow T$	$\text{First}(*) = \{*\}$	
$T \rightarrow T * F$	$\text{First}(\text{id}) = \{\text{id}\}$	
$T \rightarrow F$	$\text{First}(E) = \{(\text{, id}\}$	
$F \rightarrow (E)$	$\text{First}(T) = \{(\text{, id}\}$	
$F \rightarrow \text{id}$	$\text{First}(F) = \{(\text{, id}\}$	

Aquí miramos siempre el símbolo N a la izquierda de la regla.

Cuestión particular en $\text{First}(S)$



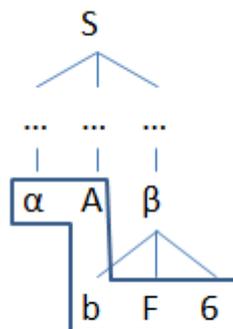
Follow

Conjunto de símbolos terminales que pueden ir a continuación del no terminal en alguna fórmula válida del lenguaje:

$$\text{Follow}(A) = \{\alpha \in T / S \rightarrow^* \alpha A \beta\} \quad A \in N$$

Aplicaremos una serie de reglas:

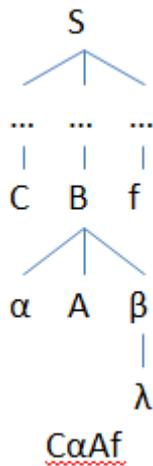
- $\$ \in \text{Follow}(S)$ en cualquier caso.
- Si $\exists B \rightarrow \alpha A \beta \rightarrow \text{First}(\beta) \in \text{Follow}(A)$



**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**





Aplicando estas tres reglas podemos calcular el follow de cualquier no terminal. Solo funciona con no terminales. En este caso siempre se buscan las reglas donde el símbolo N aparezca a la derecha de la regla.

Ejemplo

Dada la gramática de la página 33:

Follow (E) = { $\$, +,)$ } Si a la derecha del no terminal aparece un terminal este se añade directamente. $\$$ se añade por ser axioma siempre.

Follow (T) = { $\$, +,)$, *} Se añade el Follow(E) ya que T puede no tener nada a su derecha y entonces se añade al Follow(T) el Follow del padre.

Follow (F) = { $\$, +,)$, *S} Se añade el Follow(T) por el mismo motivo.

Tabla de ACCIÓN

El estado S_i se obtiene de I_i teniendo en cuenta que:

- Si $A \rightarrow \alpha \cdot a \beta \in I_i \wedge \text{Goto}(I_i, a) = I_j \rightarrow \text{Acción}(i, a) = \text{desplazar e ir a } j$
punto + T



- Si $A \rightarrow \alpha \cdot \in I_i \rightarrow \text{Acción}(i, a) = \text{reducir } (A \rightarrow \alpha)$
 $\forall a \in \text{Follow}(A)$

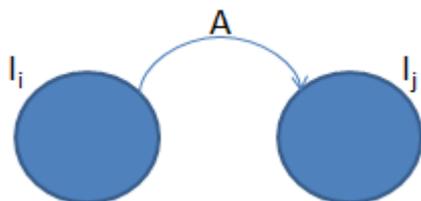
**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

Tablas de GOTO

$Goto(I_i, A) = I_j \rightarrow GOTO(i, A) = j$



$S' \rightarrow \cdot S \in$ Estado inicial I_0

Al construir la tabla de decisión en cada estado pueden pasar varias cosas:

- Aparece una sola acción \rightarrow Bien. Se coloca sin ningún problema.
- No aparece nada \rightarrow Bien. Indica que es un lugar vacío, si en la pila se llega a ese lugar se detectará un error.
- Aparecen dos o más acciones \rightarrow Mal. Conflicto. Significa que la gramática es ambigua y tenemos un problema.

Ejemplo

Usando el autómata de prefijos viables de la página 32.

Tenemos tantos estados como nodos en el autómata.

estado	ACCIÓN						GOTO		
	id	+	*	()	\$	E	T	F
S_0	D5				D4		1	2	3
S_1		D6				ACEPTAR			
S_2		R2	D7		R2	R2			
S_3		R4	R4		R4	R4			
S_4	D5				D4		8	2	3
S_5		R6	R6		R6	R6			
S_6	D5				D4			9	3
S_7	D5				D4				10
S_8		D6			D11				
S_9		R1	D7		R1	R1			

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

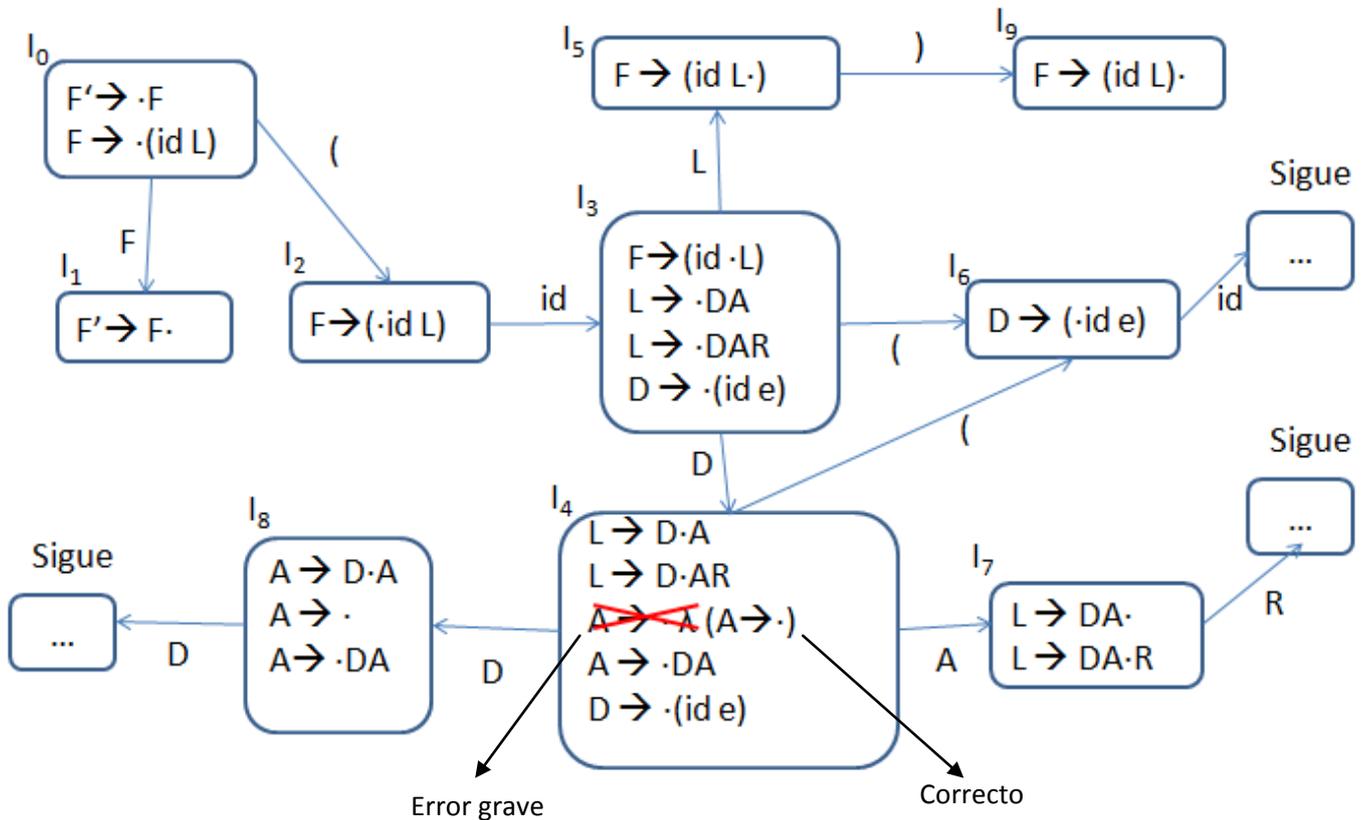
Cuando encontramos un ítem con punto al final reducimos por la regla donde esté ese ítem en los Follow del símbolo a la izquierda de la regla. El número junto a r indica el número de regla.

Ejercicio de examen

Dada la gramática:

- $F \rightarrow (id L)$
- $L \rightarrow DA / DAR$
- $A \rightarrow \lambda / DA$
- $D \rightarrow (id e)$
- $R \rightarrow rest A$

Primero aumentamos la gramática:
 $F' \rightarrow F$



Ejercicio

$S \rightarrow SA / b$ Construir autómata y tabla de decisión.
 $A \rightarrow Aa / Ab / B$

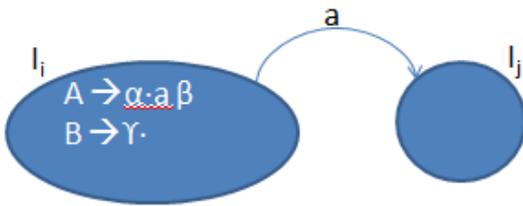
CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70



Conflictos LR

Reducción/Desplazamiento

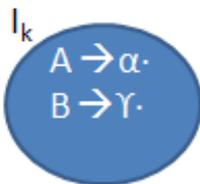


Acción[I_i , a] = desplazar e ir I_j
 Acción[I_i , Follow(B)] = Reducir $B \rightarrow \gamma$

Pero, ¿qué sucede si $a \in \text{Follow}(B)$?

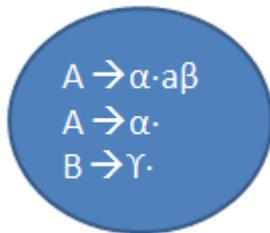
Tendremos dos acciones distintas en un mismo lugar. Eso no es posible, es un conflicto de tipo Reducción/Desplazamiento.

Reducción/Reducción



$\text{Follow}(A) \cap \text{Follow}(B) \neq \emptyset \rightarrow$ Conflicto Reducción/Reducción.

Hay que tener cuidado al aplicar las condiciones:



Si decimos que:

$\{a\} \cap \text{Follow}(A) \cap \text{Follow}(B) = \emptyset \rightarrow$ No hay Conflicto.

Puede parecer correcto, pero **NO**, con esta condición solo vemos si existen conflictos de tres posibilidades, pero no si hay de dos (que es lo más común). Para ello habría que desgranar esa condición en tres:

- $\{a\} \notin \text{Follow}(A)$
 - $\{a\} \notin \text{Follow}(B)$
 - $\text{Follow}(A) \cap \text{Follow}(B) \neq \emptyset$
- } No hay conflictos

Se recomienda encontrar los conflictos antes de hacer la tabla de decisión, para ahorrar trabajo y coste, y verificar las condiciones tras el autómata de prefijos viables.

Cartagena99

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

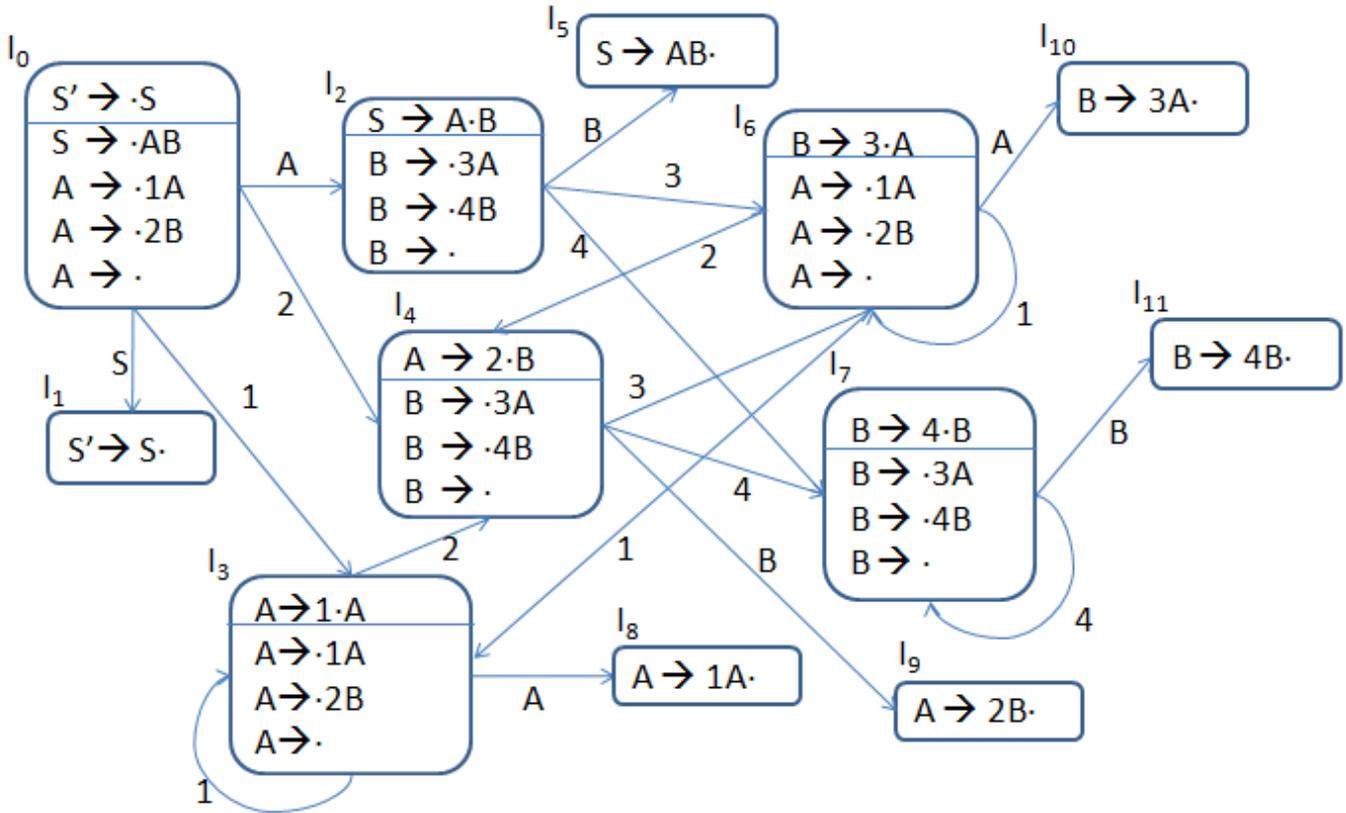
ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70

Ejercicio de examen

Dada la siguiente gramática:

$S \rightarrow AB$
 $A \rightarrow 1A / 2B / \lambda$
 $B \rightarrow 3A / 4B / \lambda$

Construimos la gramática aumentada:
 $S' \rightarrow \cdot S$



First (S) = {1, 2, 3, 4, λ }

Follow (S) = { $\$$ }

First (A) = {1, 2, λ }

Follow (A) = {3, 4, $\$$ }

First (B) = {3, 4, λ }

Follow (B) = {3, 4, $\$$ }

Susceptibles de problemas

$I_0 \rightarrow$ Red/desp

$I_3 \rightarrow$ Red/Desp

$I_6 \rightarrow$ Red/Desp

$I_2 \rightarrow$ Red/desp

$I_4 \rightarrow$ Red/Desp

$I_7 \rightarrow$ Red/Desp

De los cuales darán error $\rightarrow \{I_2, I_3, I_4\}$

En el análisis LR no existen conflictos de desplazamiento/desplazamiento.

Cartagena99

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70

Análisis sintáctico descendente

Condición necesaria

Para que una gramática pueda analizarse por método descendente es necesario que no presente recursividad por la izquierda (explicado en la [página 6](#)) y que esté factorizada por la izquierda.

Factorización por la izquierda

Si para un mismo no terminal existen dos reglas que empiezan por el mismo símbolo esa gramática no está factorizada por la izquierda:

$S \rightarrow AB$
 $A \rightarrow aC / aD \leftarrow$ Dos producciones empiezan por 'a' NO está factorizada.

Si factorizamos:

$S \rightarrow AB$
 $A \rightarrow aA'$ Similar al concepto de sacar factor común.
 $A' \rightarrow C / D$

Análisis sintáctico descendente - Predictivo recursivo

A la hora de diseñar los procesos de cada no terminal A:

1. Decidir que regla de A aplicar
 - a. Si siguiente token $\in \text{First}(\alpha) \Rightarrow$ aplicar $A \rightarrow \alpha$
 - b. Si siguiente token $\notin \text{First}(\alpha / A \rightarrow \alpha \in G) \Rightarrow$ aplicar $A \rightarrow \lambda$ (Si existe)
 - c. Si siguiente token $\in \text{First}(\alpha) \wedge \in \text{First}(B) A \rightarrow \alpha, A \rightarrow B \in G \Rightarrow$ ERROR
(Debe ser determinista)
2. Para cada símbolo de la derecha
 - a. No terminal \rightarrow Llamada a su proceso.
 - b. Terminal \rightarrow Debe coincidir con el siguiente token.

Ejemplo

$E \rightarrow E+T / T$
 $T \rightarrow T * F / F$
 $F \rightarrow (E) / id$

Primero eliminamos recursividad por la izquierda:

$E \rightarrow TE'$
 $E' \rightarrow +TE' / \lambda$
 $T \rightarrow FT'$
 $T' \rightarrow *FT' / \lambda$
 $F \rightarrow (E) / id$

Sin problemas de factorización.



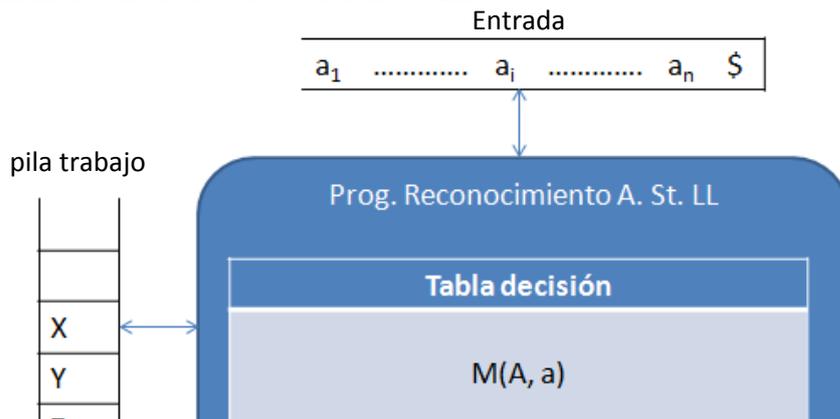
CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Programa Analizador sintáctico descendente PR

```
leer siguiente token (a);
  E;
End;
Proc E;
  T;
  E';
End;
Proc E';
  If a = First(+TE') then
    Begin
      leer siguiente token(a);
      T;
      E';
    End;
  Else  $\emptyset$ ; //Aplicación de E'  $\rightarrow \lambda$ 
End;
//El procedimiento T' sería similar con a=*
Proc F;
  If a = ( then
    leer siguiente token(a);
    E;
    leer siguiente token(a);
    if a = ) leer siguiente token(a);
    Else ERROR;
    End;
  Else if a=id then leer siguiente token(a);
  Else ERROR;
  End;
End;
```

Análisis sintáctico descendente - LL



**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

Si $CimaPila \in N \rightarrow Consultar M[CimaPila, siguiente token] = \emptyset$
 $CimaPila \rightarrow XYZ$ Sustituir $CimaPila$ por derecha de la regla.

Construcción tablas decisión LL

$\forall A \rightarrow \alpha \in G$

- Si $t \in First(\alpha) \rightarrow M(A, t) = A \rightarrow \alpha$ ($t \in T$)
- Si $\lambda \in First(\alpha) \rightarrow M(A, t) = A \rightarrow \alpha$ $\forall t \in Follow(A)$

Ejemplo

First (E) = { (, id }	Follow (E) = { }, \$ }	$E \rightarrow TE'$
First (E') = { +, λ }	Follow (E') = { }, \$ }	First (TE') = { (, id }
First (T) = { (, id }	Follow (T) = { +,), \$ }	Se hace el First de cada producción de las reglas y esta regla se sitúa en la table bajo el símbolo que sea su First.
First (T') = { *, λ }	Follow (T') = { +,), \$ }	
First (F) = { (, id }	Follow (F) = { *, +,), \$ }	

	+	*	()	id	\$
E			$E \rightarrow TE'$		$E \rightarrow TE'$	
E'	$E' \rightarrow +TE'$			$E' \rightarrow \lambda$		$E' \rightarrow \lambda$
T			$T \rightarrow FT'$		$T \rightarrow FT'$	
T'	$T' \rightarrow \lambda$	$T' \rightarrow *FT'$		$T' \rightarrow \lambda$		$T' \rightarrow \lambda$
F			$F \rightarrow (E)$		$F \rightarrow id$	

Ejercicio

$F \rightarrow (id A)$
 $A \rightarrow P O R$
 $O \rightarrow P O / \text{null}$
 $R \rightarrow \text{rest } O / \lambda$
 $P \rightarrow (id O)$

Crear el analizador sintáctico LL.

Condición LL

- Si $A \rightarrow \alpha_1, A \rightarrow \alpha_2 \in G \Rightarrow First(\alpha_1) \cap First(\alpha_2) = \emptyset$ Es decir, que no tengan nada en común.
- Si $A \rightarrow \alpha_1, A \rightarrow \alpha_2 \in G$ y $\lambda \in First(\alpha_1) \Rightarrow Follow(A) \cap First(\alpha_2) = \emptyset$
Pertenece a una de las reglas.

Si existen n producciones de A debe examinarse la condición con todas las parejas de

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70**

Cartagena99

Si $\lambda \in \text{First}(\alpha_1)$ (o a una de ellas, solo una) debería examinarse:

$$\text{Follow}(A) \cap \text{First}(\alpha_2) = \emptyset$$

...

$$\text{Follow}(A) \cap \text{First}(\alpha_n) = \emptyset$$

Ejemplo

F \rightarrow (id A)

A \rightarrow POR

O \rightarrow PO / null

R \rightarrow rest O / λ

P \rightarrow (id O)

First(F) = { (}	Follow(F) = { \$ }
First(A) = { (}	Follow(A) = {) }
First(P) = { (}	Follow(O) = { , rest }
First(O) = { (, null }	Follow(R) = {) }
First(R) = { rest, λ }	Follow(P) = { (, null }

Aplicamos la condición LL.

Las únicas susceptibles serían O y R por tener más de una producción, entonces:

$$\text{¿ } \text{First}(PO) \cap \text{First}(\text{null}) = \emptyset ?$$

$$\{ (\} \cap \{ \text{null} \} = \emptyset \rightarrow \text{La O no da problemas.}$$

$$\text{¿ } \text{First}(\text{rest O}) \cap \text{First}(\lambda) = \emptyset ?$$

$\{ \text{rest} \} \cap \{ \lambda \} = \emptyset \rightarrow$ La primera condición es correcta pero $\text{First}(\lambda) = \lambda$ así que debemos examinar la segunda condición.

$$\text{¿ } \text{First}(\text{rest O}) \cap \text{Follow}(R) = \emptyset ?$$

$$\{ \text{rest} \} \cap \{) \} = \emptyset \rightarrow \text{La R no da problemas}$$

La pregunta es, si incluimos la regla $P \rightarrow \lambda$, ¿Qué cambia?

En este caso $\text{First}(P) = \{ (, \lambda \}$ entonces al poder anularse $P \rightarrow \text{First}(A) = \{ (, \text{null} \}$

y al examinar la condición:

$$\text{¿ } \text{First}(PO) \cap \text{First}(\text{null}) = \emptyset ?$$

$$\{ (, \text{null} \} \cap \{ \text{null} \} \neq \emptyset$$

y, además habría que preguntarse:

$$\text{¿ } \text{First}(\text{id O}) \cap \text{Follow}(P) = \emptyset ?$$

$$\{ (\} \cap \{ (, \text{null} \} \neq \emptyset$$

No cumpliéndose entonces la condición LL.

Ejemplo de examen

BLOCKE \rightarrow (block SENT REST BLOCK)

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

	()	block	return	id	if	while	for	\$
BLOQUE	BLOQUE → (...								
REST-BLOCK		REST-BLOCK → λ		REST-BLOCK → return	REST-BLOCK → SENT...	REST-BLOCK → SENT...	REST-BLOCK → SENT...	REST-BLOCK → SENT...	
SENT				SENT → SENT-ASIG SENT → EXPRESION	SENT → SENT-CONTROL	SENT → SENT-CONTROL	SENT → SENT-CONTROL		
SENT-ASIG									
EXPRESION									
SENT-CONTROL									

En (SENT, id) hay dos reglas en la misma casilla, por tanto, esta gramática no cumple la condición LL y por ello no permitiría el análisis LL.

First(BLOQUE) = { (}
 First(REST-BLOCK) = { return, λ, id, if, while, for }
 First(SENT) = { id, if, while, for }

Follow(BLOQUE) = { \$ }
 Follow(REST-BLOCK) = {) }
 Follow(SENT) = { return, id, if, while, for }

Ejercicio

S → procedure division . [DECLARACIONES] {CUERPO}
 DECLARACIONES → declaratives . {STRUCT use PARRAFO} end declaratives
 CUERPO → id section . {PARRAFO}
 STRUCT → number . SENTENCE
 SENTENCE → goto id / evaluate id

Esta gramática está en formato extendido, donde lo que está entre [] (corchetes) significa que puede aparecer 1 o ninguna vez y lo que está entre {} (llaves) 0 ó n veces. Las palabras en mayúsculas son No Terminales y en minúscula Terminales.

Se trata de hacer el algoritmo predictivo-recursivo para esta gramática.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP: 689 45 44 70

Cartagena99

Análisis sintáctico descendente por tablas

Dada la gramática de la [página 42](#) y la palabra:

(id (id null) null)

Creada a partir de esa gramática el análisis por tablas es el siguiente:

PILA	Cadena	Regla
\$ F	(id (id null) null)	$F \rightarrow (id A)$
\$) A id	id (id null) null)	
\$) A id	(id null) null)	
\$) A	(id null) null)	$A \rightarrow POR$
\$) R O P	(id null) null)	$P \rightarrow (id O)$
\$) R O) O id	id null) null)	
\$) R O) O id	null) null)	
\$) R O) O	null) null)	$O \rightarrow null$
\$) R O) null	null)	
\$) R O	null)	$O \rightarrow null$
\$) R null)	
\$) R		$R \rightarrow \lambda$
\$	-	

La producción de cada regla se coloca en orden inverso en la pila { (id A) →) A id (} para que en la cabeza de la pila (que crece hacia la derecha) esté el primer símbolo de la producción.

Si el símbolo de la pila coincide con el token dado, vamos por el buen camino, el análisis está siendo correcto. Si al contrario no coinciden, es un error de la cadena.

Ejemplo de análisis predictivo-recursivo

Comprobar_token (T:Token)

BEGIN

```

    If t=sgte_token;
    Then leer_sgte_token (sgte_token);
    Else error;

```

END

PROGRAM Ejemplo Desc_Rec;

```

    Leer_sgte_token (Sgte_token);
    BLOQUE;

```

END

Procedure BLOQUE;

BEGIN

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

```

        Else if (sgte_token == id) Then
            BEGIN //Aquí habría un conflicto
porque pueden ser dos reglas.
                Comprobar_token(id);
                If(sgte_token == op-asig) Then
                    BEGIN
                        Comprobar_token(op_asig);
                        Comprobar_token(id);
                    END
                Else
                    BEGIN
                        Comprobar_token(op_arit);
                        Comprobar_token(id);
                    END
                END
            END
        END
    Procedure REST-BLOCK;
    BEGIN
        If(sgte_token ∈ {id, if, while, for) Then
            BEGIN
                SENT;
                REST-BLOCK;
            END
        Else if (sgte_token == result) Then
            Comprobar_token(return);
        Else //if sgte_token ∈ Follow(REST-BLOCK)
    END

```

Una solución para eliminar el conflicto sería cambiar la gramática por una equivalente:

SENT → S-CONTROL / id S-AUX

S-AUX → Resto_asig / Resto_expr
 op_asig id / op_arit id

Así la gramática volvería a ser LL

Análisis semántico

En el análisis semántico se utilizará una gramática de contexto libre (tipo 2) pero con un añadido denominado gramática de atributos. Estos atributos que añadiremos a los no terminales nos darán información sobre su contenido semántico.

Estos atributos se otorgan a los símbolos mediante las **reglas semánticas**:

$S \rightarrow \alpha \quad \{ \text{atributos} \}$

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70**

Cartagena99

Traducción dirigida por la sintaxis

Se presenta con dos notaciones diferentes:

- Definición dirigida por la sintaxis (DDS) → Notación de alto nivel con menos detalles.
- Esquema de traducción (EDT) → Notación de bajo nivel con más detalles.

Lo que estamos trabajando es el diseño del analizador semántico, lo que quiere decir que si trabajamos con DDS nos tendremos que preocupar de menos cosas pero será cosas que tendremos que definir más tarde.

Todo lo contrario es trabajar con EDT, durante el diseño da más trabajo pues te obliga a trabajar con más detalles pero allana el camino para más tarde.

Atributos

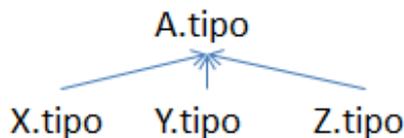
Se aplican sobre los no terminales:

A → XYZ donde puedo definir un atributo tipo (por ejemplo):

$$A.\text{tipo} = f(x.\text{tipo}, y.\text{tipo}, z.\text{tipo})$$

Los atributos pueden ser de dos tipos:

- **Sintetizados:** Se calculan, exclusivamente, a partir de los atributos del lado derecho de la regla. A partir de los hijos.



Como es el caso del ejemplo.

- **Heredados:** Cualquier atributo no sintetizado. Se calculan a partir de cualquier tipo de atributo, no exclusivamente de los hijos sino también de los hermanos y el padre.

$$X.\text{Tipo} = f(Y.\text{tipo}) \quad X.\text{Posición} = A.\text{posición}$$

Ejemplo

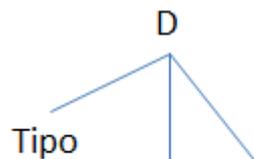
Dada la frase:

int edad;

con la gramática:

D → Tipo id;

Árbol sintáctico:



Hasta ahora solo nos fijábamos en si la estructura es correcta, ahora debemos definir que el tipo es entero(int).

Tenemos que unir de alguna manera el tipo ese id, la única manera es con un atributo que recorra las

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

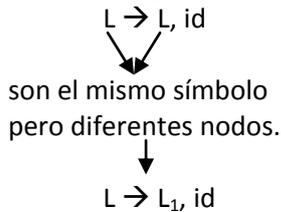
ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

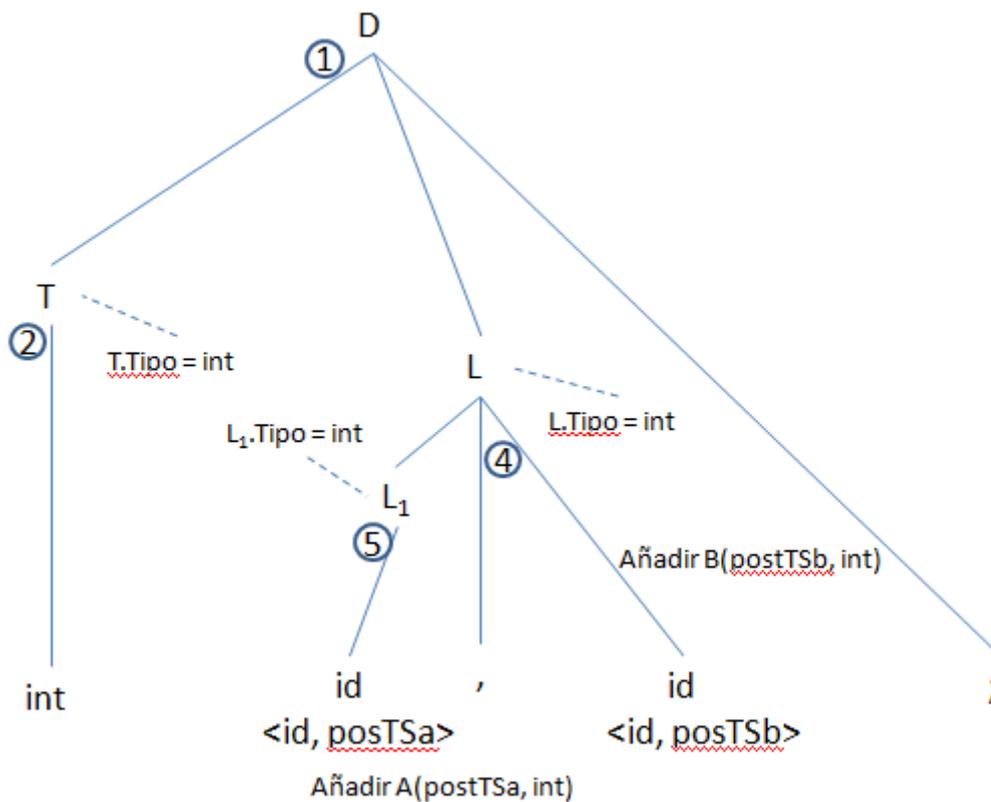
Veámoslo con un ejemplo:

$D \rightarrow T L ;$ {L.tipo = T.tipo}
 $T \rightarrow \text{int}$ {T.tipo = int}
 $T \rightarrow \text{float}$ {T.tipo = float}
 $L \rightarrow L_1, \text{id}$ {L₁.tipo = L.tipo; Añade tipo TS(id.postTS, L.tipo)}
 $L \rightarrow \text{id}$ {Añade tipo B(id.posB, L.tipo)}

Como se puede observar en la regla 4, diferenciamos (mediante subíndices) los símbolos no terminales iguales en una misma regla:



Dada la palabra *int a, b;*



Nos faltaría saber muchas cosas aún, la primera de ellas: ¿Qué orden seguimos? En qué momento se hace cada cosa difiere mucho según se utilice DDS o EDT.



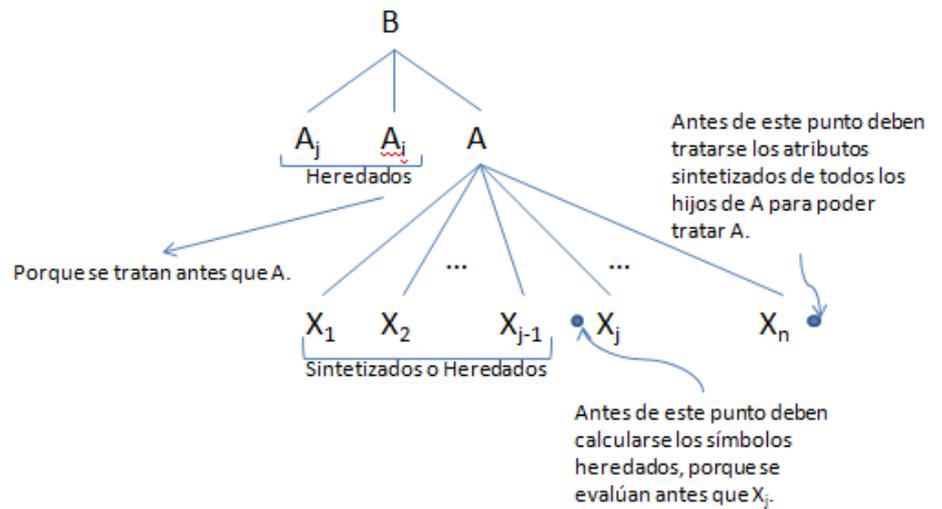
CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70

La idea que se persigue es que, cuando llegue el momento de expandir un nodo ya tenemos toda la información relevante sobre él.

Como consecuencia, deben cumplirse las siguientes condiciones cuando hay atributos sintetizados y heredados:

- Un atributo heredado para un símbolo del lado derecho de la regla debe calcularse antes de que se trate dicho símbolo.
- Un atributo sintetizado para el no terminal de la izquierda "sólo" puede calcularse una vez tratados todos sus hijos.



Formas de representación

$D \rightarrow T L ; \quad L.tipo = T.tipo$
 $T \rightarrow int \quad T.tipo = int$
 $T \rightarrow float \quad T.tipo = float$
 $L \rightarrow L_1, id \quad L_1.tipo = L.tipo; \text{ Añade tipo TS}(id.PosTS, L.tipo)$
 $L \rightarrow id \quad \text{Añade tipoTS}(id.PosTS, L.tipo)$

DDS \rightarrow Tenemos reglas y acciones semánticas pero no nos preocupamos de cuando se implementa cada acción semántica.

En EDT sí debemos preocuparnos de este detalle. Tenemos que utilizar la estrategia de atributos por la izquierda, y resulta:

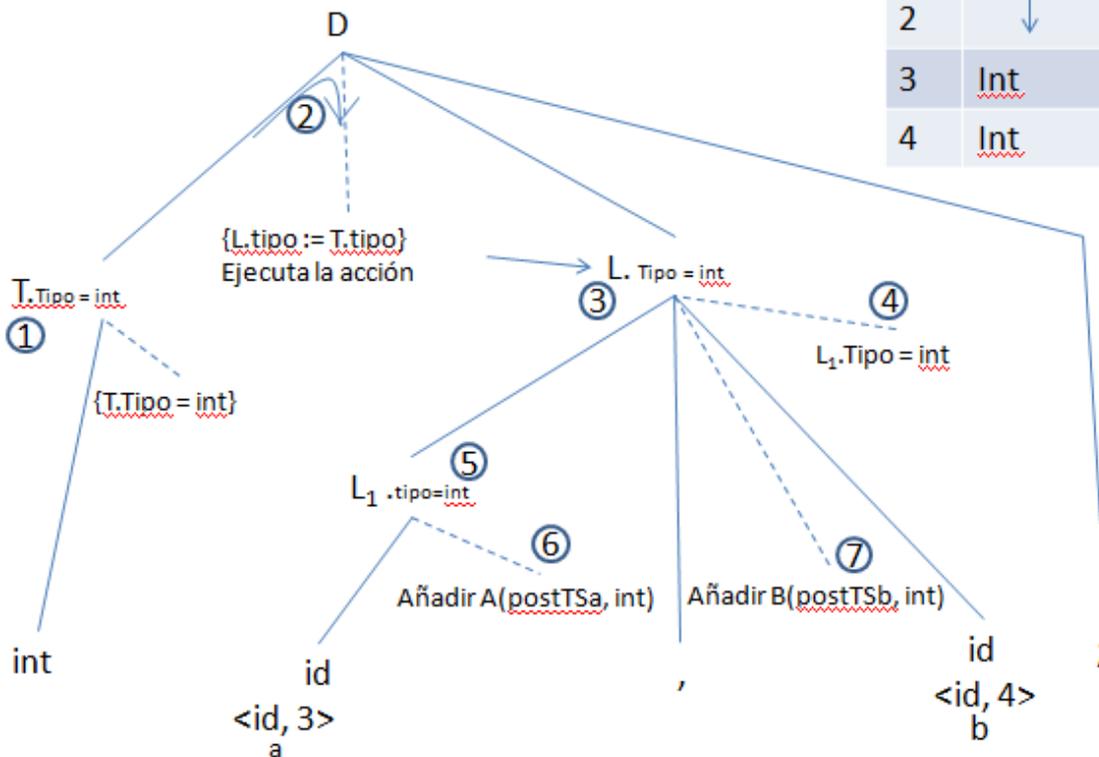
$D \rightarrow T\{ L.tipo = T.tipo\} L ;$
 $T \rightarrow int \{T.tipo = int\}$
 $T \rightarrow float \{T.tipo = float\}$
 $L \rightarrow \{ L_1.tipo = L.tipo \} L_1, id \{ \text{Añade tipo TS}(id.PosTS, L.tipo) \}$
 $L \rightarrow id \{ \text{Añade tipoTS}(id.PosTS, L.tipo) \}$

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

Id.	Lexema	Tipo
1		
2		
3	<u>Int</u>	a ⑥
4	<u>Int</u>	b ⑦



Aunque en este caso nos hemos permitido hacer un análisis descendente aunque no sería del todo posible si la frase fuese más larga.

Comprobaciones semánticas

Sistemas de tipos

Orientado a la comprobación de tipos. No es más que un conjunto de reglas cuyo propósito es asignar una expresión de tipo a las distintas estructuras del texto fuente, en el caso que nos ocupa a las estructuras del lenguaje de programación.

El comprobador de tipos (subconjunto muy grande del analizador semántico, incluso podemos generalizar que es el analizador por completo) es la implementación de sistemas de tipos.

Expresión de tipos

Denota el tipo de una construcción del texto fuente. Existen dos tipos.

Tipos básicos

Todos aquellos tipos definidos dentro del propio lenguaje. Aunque existen tres tipos básicos especiales:

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Tipos contruidos

Se basan en la aplicación de un constructor a uno o más tipos básicos o contruidos.

Constructores de tipos

Veamos los casos más comunes:

Vectores

Si T es expresión de tipo \rightarrow array(I, T) es expresión de tipo (donde I son los índices).

Ejemplo

char V[10] \rightarrow array(1..10, char) se asigna a V.

Producto cartesiano

Si T₁, T₂ son expresiones de tipo \rightarrow T₁X T₂ es expresión de tipo.

Este constructor nace más como una necesidad del compilador que como expresión del lenguaje. Luego veremos por qué.

Registro

Si T₁,...,T_n son expresiones de tipo \rightarrow record (T₁X T₂x...xT_n) es expresión de tipo

Ejemplo

Struct R

```
{ int a;           $\rightarrow$  record(int x array(1..5, real))  $\rightarrow$  R
  float b[5]      ó 0..4 esto dependen del lenguaje.
}
```

Punteros

Si T es expresión de tipo \rightarrow pointer(T) es expresión de tipo.

Funciones

int ejemplo (float, char) \rightarrow float x char --> int es expresión de tipo que se asigna a ejemplo.

Ejemplo

P \rightarrow D ; S **Declaración ; sentencias**

D \rightarrow D₁; D₂ / id:T

T \rightarrow **básicos** char / integer / boolean / **construidos** **puntero** \uparrow T₁ / **array** array[num] of T₁

S \rightarrow S₁ ; S₂ / **asig** id:=E / **if-then** if E then S₁ **Sentencias**

E \rightarrow car / num / true / false / id / E₁ mod E₂ / **operaciones** $\sqrt{E_1[E_2]}$ / **modulo** $E_1 \uparrow$ / **puntero** E_1 and E_2 / **lógica** E_1 op.rel E_2 / **general**

Recordemos que los subíndices sirven para diferenciar los no terminales con mismo nombre

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

$T \rightarrow \text{boolean} \quad \{T.\text{tipo} = \text{boolean}\}$
 $T \rightarrow \uparrow T_1 \quad \{T.\text{tipo} = \text{pointer}(T_1.\text{tipo})\}$ **Como se ve, tipo construido.**
 $T \rightarrow \text{array}[\text{num}] \text{ of } T_1 \quad \{T.\text{tipo} = \text{array}(1..\text{num}.\text{valor}, T_1.\text{tipo})\}$
 $S \rightarrow S_1 ; S_2 \quad \{S.\text{tipo} := (\text{If } S_1.\text{tipo} = \text{tipo_OK and } S_2.\text{tipo} = \text{tipo_OK then tipo_OK Else tipo_error})\}$
 $S \rightarrow \text{id} := E \quad \{\text{id}.\text{tipo} = \text{BuscaTipoTS}(\text{id}.\text{entrada}) ; S.\text{tipo} := (\text{If id.tipo} = E.\text{tipo Then tipo_OK Else tipo_error})\}$
 $S \rightarrow \text{If } E \text{ then } S_1 \quad \{S.\text{tipo} := (\text{If } E.\text{tipo} = \text{Boolean and } S_1.\text{tipo} = \text{tipo_OK Then tipo_OK Else tipo_error})\}$
 $E \rightarrow \text{car} \quad \{E.\text{tipo} = \text{char}\}$
 $E \rightarrow \text{num} \quad \{E.\text{tipo} = \text{integer}\}$ **(Solo hay expresiones de tipo entero en el lenguaje)**
 $E \rightarrow \text{true} \quad \{E.\text{tipo} = \text{boolean}\}$
 $E \rightarrow \text{false} \quad \{E.\text{tipo} = \text{boolean}\}$
 $E \rightarrow \text{id} \quad \{E.\text{tipo} = \text{buscaTipoTS}(\text{id}.\text{entrada})\}$
 $E \rightarrow E_1 \text{ mod } E_2 \quad \{E.\text{tipo} := (\text{If } E_1.\text{tipo} = \text{integer and } E_2.\text{tipo} = \text{integer Then integer Else tipo_error})\}$
 $E \rightarrow E_1[E_2] \quad \{E.\text{tipo} := (\text{If } E_2.\text{tipo} = \text{integer and } E_1.\text{tipo} = \text{array}(i,t) \text{ Then } t \text{ Else tipo_error})\}$
(Este es un caso muy sencillo, en un caso real habría que hacer más operaciones)
 $E \rightarrow E_1 \uparrow \quad \{E.\text{tipo} := (\text{If } E_1.\text{tipo} = \text{pointer}(t) \text{ Then } t \text{ Else tipo_error})\}$
 $E \rightarrow E_1 \text{ and } E_2 \quad \{E.\text{tipo} := (\text{If } E_1.\text{tipo} = \text{boolean and } E_2.\text{tipo} = \text{boolean Then boolean Else tipo_error})\}$
 $E \rightarrow E_1 \text{ op. rel } E_2 \quad \{E.\text{tipo} := (\text{If } E_1.\text{tipo} = \text{integer and } E_2.\text{tipo} = \text{integer Then boolean Else tipo_error})\}$

Podemos añadir la regla para sumar:

$E \rightarrow E_1 + E_2 \quad \{E.\text{tipo} := (\text{If } E_1.\text{tipo} = \text{integer and } E_2.\text{tipo} = \text{integer Then integer Else tipo_error})\}$

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70**

Cartagena99

Este es un ejemplo con un lenguaje sin conversión automático de tipos.

Con conversión automática de tipos sería:

```
E → E1 + E2    {E.tipo := (If E1.tipo = int and E2.tipo = int Then int
                    If E1.tipo = int and E2.tipo = float Then float
                    If E1.tipo = float and E2.tipo = int Then float
                    If E1.tipo = float and E2.tipo = float Then float
                    Else tipo_error)}
```

Ejercicio

```
C : char;
i : integer;
i : c mod i mod 3
```

Ejercicio resuelto

```
P → DP / SP / λ
D → function id (L) : T ; begin P end
D → T id;
L → T : id / L ; L
T → integer / real
S → id := E;
E → id (A) / id
A → E / A, A
```

Lo primero que se debe hacer es iniciar todas las pilas y variables, esto suele hacerse en la primera regla, antes de la primera producción, pero en este caso, la primera regla no siempre se utiliza solo al principio. Por ello hay que colocar las sentencias iniciales en otro lugar. Hay dos soluciones:

1. Inicializar todo antes de comenzar a trabajar. Dentro del código del analizador semántico pero no dentro de la gramática.

Lenguaje paso de parámetros por valor. Sin conversión automática de tipos.

Construir EDT, detallando accesos a TS incluyendo construcción y destrucción y teniendo en cuenta que el analizador léxico no introduce nada en la tabla de símbolos.

EDT

- Comprobaciones de tipo.
- Comprobaciones de que las sentencias son correctas.
- Manejar TS.
- Insertar id en TS, y la información asociada.
 - ¿Dónde estará cada id en tipo de ejecución?

En este ejercicio se permiten funciones dentro de funciones. Por tanto se crearán tablas de símbolos anidadas para cada una de ellas. Para poder manejar esto habrá que crear una pila de tablas de símbolos.

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

desplazamiento de cada elemento en una TS.

```

P' → { PilaTS := nuevaPila()
        PilaDesp := nuevaPila()
        push (PilaTS, crearTS())
        push (PilaDesp, 0) } P

```

P → DP / SP / λ (No vamos a colocar ninguna acción en este punto, aunque podríamos verificar que no se comenten errores y cosas así).

```

D → function id { If buscarTS (cima (PilaTS), id.lexema) ≠ null
                  Then Error ("Función ya declarada")
                  Else   id.entrada := insertarTS (cima (PilaTS), id.lexema)
                          push (pilaTS, crearTS())
                          push (pilaDesp, 0) }

```

(L) : T { insertarTipoTS (L.tipo → T.tipo), id.entrada¹ } (L.tipo lo gestionaremos en sus reglas de producción)
; begin P end { pop (PilaTS), pop (PilaDesp) }

```

D → T id ; { If buscarTS (cima (PilaTS), id.lexema) ≠ null
              Then Error ("Parámetro ya declarada")
              Else   id.entrada := insertarTS (cima (PilaTS), id.lexema)
                      insertarTipoTS (T.tipo, id.entrada)
                      insertarDespTS (cima (PilaDesp), id.entrada)
                      cima (PilaDesp) := cima (PilaDesp) + T.ancho2 }

```

```

L → T id ; { If buscarTS (cima (PilaTS), id.lexema) ≠ null
              Then Error ("Parámetro ya declarada")
              Else   id.entrada := insertarTS (cima (PilaTS), id.lexema)
                      insertarTipoTS (T.tipo, id.entrada)
                      insertarDespTS (cima (PilaDesp), id.entrada)
                      cima (PilaDesp) := cima (PilaDesp) + T.ancho;
                      L.tipo := T.tipo }

```

```

L → L1 ; L2 { L.tipo := L1.tipo x L2.tipo }

```

```

T → integer { T.tipo = integer ; T.ancho = 2 }

```

```

T → real { T.tipo = real; T.ancho = 4 }

```

```

S → id := E { id.tipo = buscaTipoTS(id.entrada) ;
              If id.tipo = E.tipo
                Then S.tipo := tipo_OK
                Else S.tipo := tipo_error }

```

```

E → id(Λ) { If buscaTipoTS (id.lexema)3 := arg tipo → t AND arg tipo = Λ tipo

```

Notas

1. Podría parecer que esta función inserta el tipo en la nueva tabla creada, sin embargo insertaTipoTS trabaja de forma inteligente y recorre toda la pila de TS hasta encontrar el identificador correcto.
2. Asumimos que mata la cima de la pila e inserta la nueva cima.
3. buscaTipoTS recorre la pila de TS entera hasta dar con la primera coincidencia.

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70**

Cartagena99


```

begin B end { If B.tipoDev ≠ T.tipo
              Then Error("El tipo devuelto no es correcto")
              destruye(TSL); TSL = null; TS := TSG}

```

```

L → T : id { If buscaTS(TSL, id.lex) ≠ null
             Then Error("Parámetro ya declarado")
             Else   id.ent := insertaTS(TSL, id.lex)
                   insertaTipoParametro(TSL, id.ent, T.tipo)1
                   insertarDespl(TSL, id.ent, desplL)
                   desplL := desplL + 22
                   L.tipo := T.tipo }

```

```

L → L1 , L2 { L.tipo := L1.tipo x L2.tipo }

```

```

T → integer / real (Igual que en el ejercicio anterior)

```

```

B → DB1 { B.tipoDev := B1.tipoDev }

```

```

B → λ { B.tipoDev := vacio }

```

```

B → SB1 { If S.tipoDev = B1.tipoDev
            Then B.tipoDev := S.tipoDev
            Else if B1.tipoDev = vacio
            Then B.tipoDev := S.tipoDev
            Else if S.tipoDev = vacio
            Then B.tipoDev := B1.tipoDev
            Else error("Tipos devueltos erróneos")

```

```

S → return E { S.tipoDev := E.tipo
               S.tipo := tipoOK } (Aunque la última regla no se suele utilizar)

```

```

S → id := E { id.ent := busca(TS, id.lex);
              If id.ent = null
              Then   error("Variable no declarada")
                    S.tipo := tipoError
              Else if buscaTipo(TS, id.ent) ≠ E.tipo
              Then   error("Tipos erroneos en la asignación")
                    S.tipo := tipoError
              Else
                S.tipo := tipoOk
                S.tipoDev := vacio }

```

```

E → id(A) { (Como en el ejercicio anterior) }

```

Notas

1.- Utilizamos una función distinta porque normalmente hay que hacer cosas distintas.

2.- Al ser paso de parámetros por referencia el desplazamiento es simplemente guardar los bytes correspondientes a la dirección de memoria, 2 en este caso.



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70

Ejercicio

P → DS

D → T : L / D ; D

T → integer / real / boolean / array

L → id / id [numEntero] of T / L ; L

S → for id := E to E do S / Repeat S until C / id := E / S ; S

E → E + E / E x E / id / numEntero / true / false / id[E]

Hacer el esquema de traducción teniendo en cuenta:

Con convención de tipos

+ → Aritmético y lógico (OR)

x → aritmético y lógico (AND)

A.léxico ya introduce tokens en la TS.

Comprobar c.tipo = boolean



Apuntes de procesadores del lenguaje by [Pau Arlandis Martínez](#) is licensed under a [Creative Commons Reconocimiento 3.0 Unported License](#).

Cartagena99

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70