Procesadores de lenguaje
Tema 4 – Ejercicios Análisis semántico

Salvador Sánchez, Daniel Rodríguez
Departamento de Ciencias de la Computación
Universidad de Alcalá

Ejercicio 1

- A partir de gramática de abajo se pide:
  - Mostrar el árbol sintáctico decorado para la cadena de entrada "aaaa"
  - Elaborar el grafo de dependencias para la evaluación en el orden correcto de todos los atributos.
  - Indique la salida que provoca el análisis de la entrada "aaaa"

S'→S	`printf("%d",S.v);`
S → AS	S_0`.v = A.v` `A.x =` S_1`.v`
S → A	`A.x = 0` `S.v = A.v`
A → a	`A.v = A.x + 1`

Ejercicio: Binario a decimal

- Considerar la siguiente gramática que genera números binarios:

A → AB | B
B → 0 | 1

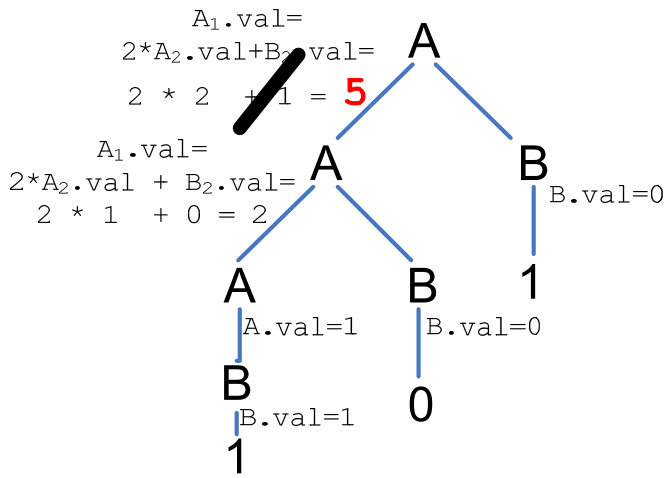
- Obtener la gramática con atributos que al analizar un número binario calcule su equivalente en decimal.
- Mostrar el árbol para la cadena de entrada `101`

Ejercicio: Binario a decimal

- Gramática con atributos

A → AB	`{A0.val = 2 * A1.val + B.val}`
A → B	`{A.val = B. val}`
B → 0	`{B.val =0}`
B → 1	`{B.val =1}`

Ejercicio: Binario a decimal



Ejercicio 3

- Considerar la siguiente gramática:

  num → dig num | dig
  dig → 0 | 1 | … | 9

- Añadirle los atributos semánticos para que calcule su valor

- Solucion:

Ejercicio octal-decimal

- Dada la gramática siguiente:
  - Dibujar las gráficas de dependencias para cada regla
  - Dibujar el árbol sintáctico para la expresión **345o**
  - Dibujar la gráfica de dependencia para el árbol sintáctico anterior y establecer una clasificación topológica para el mismo
  - Generar el código correspondiente a las reglas semánticas asociadas al análisis de la expresión anterior

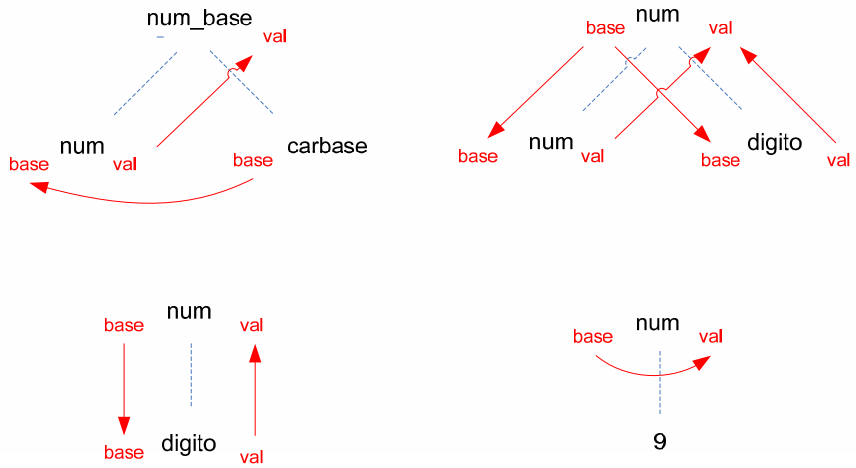
Ejercicio octal-decimal

num-base → num carbase	`num-base.val = num.val;` `num.base = carbase.base`
carbase → o	`carbase.base = 8`
carbase → d	`carbase.base = 10`

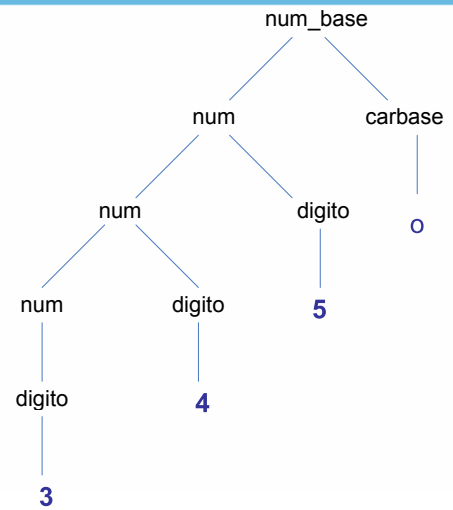
num_1 → num_2 dígito	num_1`.val =`
	`if dígito.val = error or num2.val = error`
	`then error`
	`else` num_2`.val * ` num_1`.base + dígito.val`
	num_2`.base=` num_1`.base`
	`dígito.base =` num_1`.base`

num → dígito	`num.val = dígito.val` `dígito.base = num.base`
dígito → 0	`dígito.val = 0`
...	…
dígito →7	`dígito.val = 7`
dígito → 8	`dígito.val = if dígito.base = 8 then error else 8`
dígito → 9	`dígito.val = if dígito.base = 8 then error else 9`

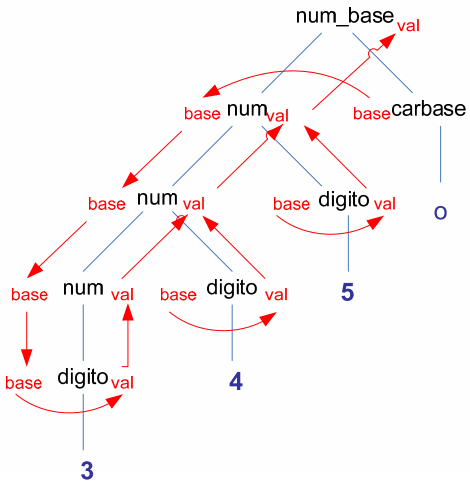
Octal-decimal. Graf. de dependencias



Octal-decimal. Árbol sintáctico 345o



Octal-decimal. Gráf. de dependencia



Octal-decimal. Código

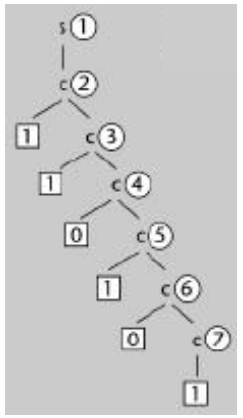
```
a1=8;  //carbase.base=8
a2=a1; //m.base=8
a3=a2; //m.base=8
a4=a3; //m.base
a5=a4; //digito.base=8
a6=3;
a7=a6;
a8= a3;
a8=4;
if (a9=error||a7==error) then a10=error else a10=a7*a3+a9;
  // a10=3*8+4=28
a11=a2; //a11=8
a12=5;
if (a12=error||a10==error) then a13=error else a13=a10*a2+a12;
 // a13=28*8+5
a13=a13;
```


- A partir de la gramática siguiente, ampliadla con **atributos sintetizados** de manera que, cuando se reduzca por S, se muestre el número de **a** y **b** que contiene una frase cualquiera del lenguaje generado por la gramática:
  - **S → (A)**
  - **A → A , D**
  - **A → D**
  - **D → a**
  - **D → b**
  - **D → (A)**

Gramática	Reglas semánticas	Comentarios
S → (A)	Muestra(A.a, A.b)	Se muestra el resultado.
A_1 → A_2 , D	A_1.a = A_2.a + D.a A_1.b = A_2.b + D.b	Es el único lugar en el que se suman los datos que provienen de A y los que provienen de D.
A → D	A.a = D.a A.b = D.b	Se pasa la información.
D → a	D.a = 1	Se inicializa el valor de *D.a*.
D → b	D.b = 1	Se inicializa el valor de *D.b*.
D → (A)	D.a = A.a D.b = A.b	Se pasa la información.

- Ampliad la gramática siguiente con **atributos sintetizados** para que obtenga el número decimal correspondiente al número binario de entrada:
  - **S → C**
  - **C → 0 C**
  - **C → 1 C**
  - **C → 0**
  - **C → 1**
- Construid el árbol sintáctico para **110101**. Numerad los nodos y hojas.
- Dad un orden de evaluación correcto. Evaluad el árbol aplicando las reglas semánticas.

Gramática	Reglas semánticas	Comentarios
S → C	mostrar (C.val)	Se muestra el número decimal.
$C_1 \rightarrow 0\ C_2$	$C_1.it = C_2.it + 1$ $C_1.val = C_2.val$	Se incrementa el contador de iteraciones. El cálculo parcial no se modifica.
$C_1 \rightarrow 1\ C_2$	$C_1.it = C_2.it + 1$ $C_1.val = C_2.val + 2^{C.it}$	Se incrementa el contador de iteraciones y se hace el cálculo parcial.
C → 0	C.val = 0 C.it = 0	Se inicializa *C.val* a 0 y el contador de iteraciones a 0.
C → 1	C.val = 1 C.it = 0	Se inicializa *C.val* a 1 y el contador de iteraciones a 0.



Procesadores de lenguaje – Tema 4: Análisis semántico
Salvador Sánchez, Daniel Rodríguez

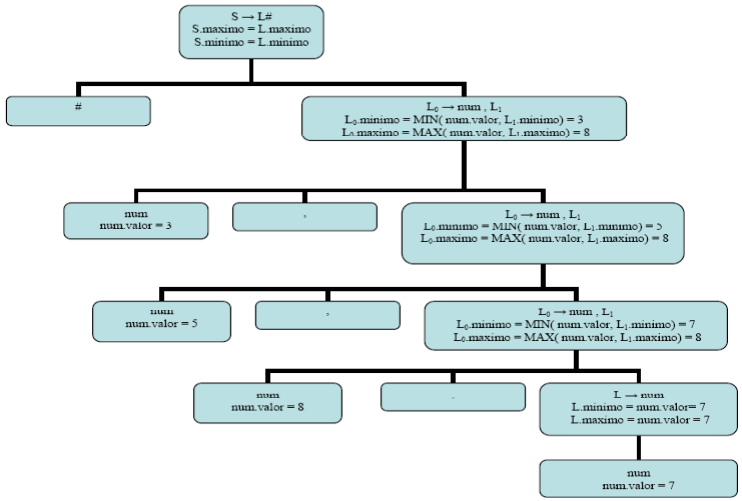
Paso	Nodo	Regla sintáctica	Acción semántica que se aplica (código parcial)
1	7	C → 1	C.val = 1 C.it = 0
2	6	$C_1 \rightarrow 0\ C_2$	$C_1.it = C_2.it + 1 = 0 + 1 = 1$ $C_1.val = C_2.val = 1$
3	5	$C_1 \rightarrow 1\ C_2$	$C_1.it = C_2.it + 1 = 1 + 1 = 2$ $C_1.val = C_2.val + 2^{C_2.it} = 1 + 4 = 5$
4	4	$C_1 \rightarrow 0\ C_2$	$C_1.it = C_2.it + 1 = 2 + 1 = 3$ $C_1.val = C_2.val = 5$
5	3	$C_1 \rightarrow 1\ C_2$	$C_1.it = C_2.it + 1 = 3 + 1 = 4$ $C_1.val = C_2.val + 2^{C_2.it} = 5 + 16 = 21$
6	2	$C_1 \rightarrow 1\ C_2$	$C_1.it = C_2.it + 1 = 4 + 1 = 5$ $C_1.val = C_2.val + 2^{C_2.it} = 21 + 32 = 53$
7	1	S → C	mostrar (53)

Procesadores de lenguaje – Tema 4: Análisis semántico
Salvador Sánchez, Daniel Rodríguez

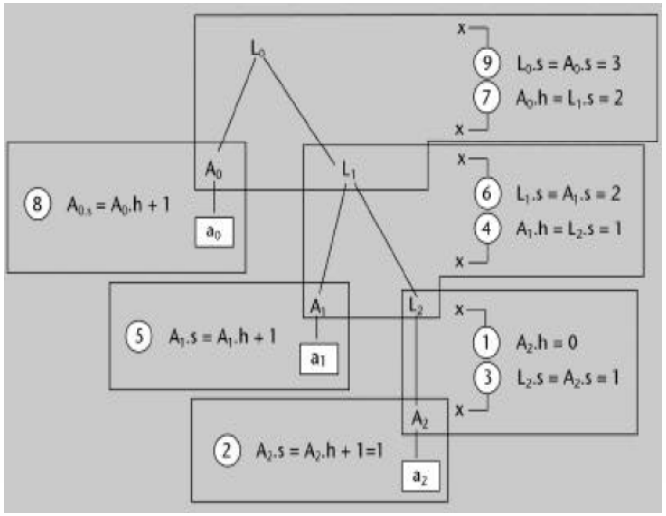
- Ampliad la gramática siguiente con **atributos sintetizados** para que calcule el valor mínimo y máximo de una lista de enteros.
  - S → L
  - L → **num** , L
  - L → **num**
- Presentad las diferentes acciones semánticas para la entrada siguiente:
  - **7, 8, 5, 3 $**

Procesadores de lenguaje – Tema 4: Análisis semántico
Salvador Sánchez, Daniel Rodríguez

S → L $	{S.max=L.max; S.min=L.min; }
L0 → num , L1	{L0.min = MIN (num.valor, L1.min); L0.max = MAX (num.valor, L1.max);}
L → num	{ L.max=num.valor; L.min=num.valor; }



Gramática	Reglas semánticas	Comentarios
$L_0 \rightarrow$ A L_1	$L_0.s = A.s$ (sintetizado) $A.h = L_1.s$ (heredado)	*A.s, L.s*: atributos sintetizados. *A.h*: atributo heretado. Finalmente, *L.s* contiene el número de *a* reconocidas.
L → A	$A.h = 0$ $L.s = A.s$ (sintetizado)	
A → a	$A.s = A.h + 1$ (sintetizado)	



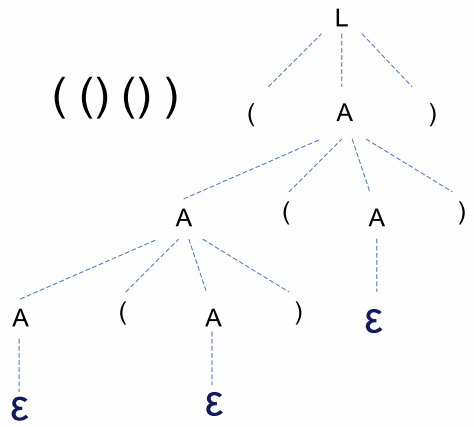
Ejercicio. Listas

- Dada la gramática:
  - lista → (lista_interna)
  - lista_interna → (lista_interna) | ε
- Es decir:
  - L → (A)
  - L → A (A) | ε
- Se pide:
  - Calcular la profundidad:
    - Ej. (() (())) → 3

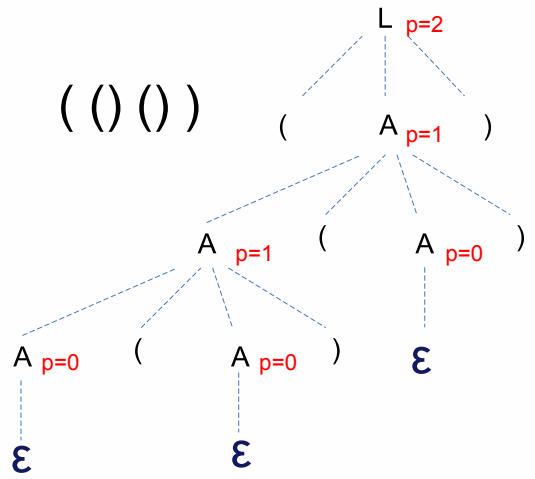
Ejercicio Listas. Profundidad

$L \rightarrow (A)$	$\{L_p = A_p + 1;\}$
$A_0 \rightarrow A_1 (A_2)$	**if** $(A_1.p > A_2.p)$ **then** $A_0.p = A_1.p$ **else** $A_0.p = A_2.p + 1;\}$
$L \rightarrow \varepsilon$	$\{A_p = 0;\}$

Ejercicio Listas. Profundidad



Ejercicio Listas. Profundidad



Ejercicio Listas. Número de listas

$L \rightarrow (A)$	A_1.listas_antes=1; L.listas_total=A.listas_despues;
$A_0 \rightarrow A_1 (A_2)$	A_1.listas_antes=A_0.listas_antes; A_2.listas_antes=A_1.listas_despues+1; A_1.listas_antes=A_2.listas_despues;
$L \rightarrow \varepsilon$	A.listas_despues=A.listas_antes;

Ejercicio tipos

- Consideremos la gramática siguiente :
  - finalExpr → expr
  - expr → expr × expr
  - expr → **(** expr **)**
  - expr → item
  - item → **INT**
  - item → **REAL**
- Aumentar la gramática con las reglas semánticas necesarias para definir el atributo TIPO del resultado final de la expresión (*finalExpr)*.
  - Los terminales INT y REAL, son valores numéricos de tipo ENTERO y REAL, respectivamente. Cuando se multiplican dos datos enteros, el resultado es entero. En cualquier otra combinación, el resultado es real.
- Representar el árbol sintáctico generado para la entrada:
- 1.72 × 3 × (3.5)

Solución (acciones semánticas)

- **a)**

finalExpr → expr { finalExpr.TIPO = expr.TIPO; }
$expr_0$ → $expr_1$ × $expr_2$
{ if ($expr_1$.TIPO == REAL || $expr_2$.TIPO == REAL)
$expr_0$.TIPO = REAL;
else
$expr_0$.TIPO = ENTERO; }
$expr_0$ → ($expr_1$) { $expr_0$.TIPO = $expr_1$.TIPO; }
expr → item { expr.TIPO = item.TIPO; }
item → INT { item.TIPO = ENTERO; }
item → REAL { item.TIPO = REAL; }

Solución (árbol)