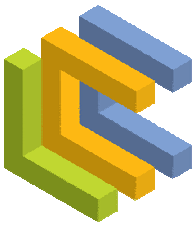
Procesadores de Lenguajes

Cap. IV
Gramáticas con atributos



LENGUAJES Y
CIENCIAS DE LA
COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

Gramáticas con atributos

- A un elemento de una gramática (terminal o no terminal) se le pueden asociar **atributos.**
- Ej. X ∈ NUT son atributos X.a , X.tipo , X.valor , X.s...
- A cada regla de producción se le pueden asociar **reglas semánticas** donde se calculan los valores de los atributos.
- Mediante reglas semánticas se puede
  - generar código
  - guardar información en la tabla de símbolos
  - emitir mensajes de error
  - realizar la comprobación de tipos
  - ...
- Ejemplo de **gramática atribuida**

$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow DIGITO$	$F.val = DIGITO.valex$

- Un atributo puede representar “cualquier cosa”:
  - Una cadena
  - Un número
  - Un tipo
  - Una dirección de memoria
  - Un puntero
  - ...
- El conjunto de atributos de X se denota por **A (x)**
- A(E) = {E.val} A(T) = {T.val} A(F) = {F.val}
  A(DIGITO) = {DIGITO.valex}
- Un atributo de un terminal (token) se denomina **“intrínseco”,** y es calculado por el lexicográfico.

- **GRAMÁTICA ATRIBUIDA**, GA= (G, A, R)
  - G gramática de contexto libre (N,T,P,S)

$A = \bigcup_{X \in NUT} A(X)$, donde $A(X) \bigcap A(Y) = \phi \ \forall X, Y \in NUT$

$R = \bigcup_{\forall p \in P} R(p)$ donde R(p) es el conjunto de Reglas semánticas asociadas a la regla de producción "p"

- **Regla Semántica**

Dada $p \equiv X_0 \rightarrow X_1 X_2 \ldots X_n$

Son de la forma:
$v = f(w_{i1}, w_{i2}, \ldots, w_{ik})$ donde

- f es una función
- $v, w_{ij} \in \bigcup_{l=0}^{n} A(X_l)$ (es decir atributos de los símbolos de p)

- **DEFINICIÓN ATRIBUIDA Ó DEFINICIÓN DIRIGIDA POR LA SINTAXIS**

Cuando las funciones de las reglas semánticas pueden tener efectos laterales: imprimir un valor, asignar un valor a una variable global (tb. símb.),

Ej.	$L \rightarrow E$ ' \n'	print (E.val)
	$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
	...	...

- **ATRIBUTO SINTETIZADO**

AS (X) = { $X.a \in A(X)$ / $\exists$ $p \equiv X \rightarrow \tau$ de forma que X.a se calcula mediante una regla semántica de p (R (p)). Es decir $X.a = f(w_1, w_2, \ldots, w_k)$
w_i atributos de los símbolos de τ - o incluso de X-}

Ej. $E.val = E_1.val + T.val$

ATRIBUTO HEREDADO

AH (X) = { X.a ∈ A (X) / ∃ p ≡ Y → α X β de forma que X.a se calcula mediante una regla semántica de p (R (p)). Es decir

$$X.a = f(w_1, w_2, \ldots, w_k)$$

w_i atributos de los símbolos de la regla "p" (Y, α, β - o incluso de X-) }

EJEMPLO

D -> T L ;	L.her = T.tipo
T -> INT	T.tipo = integer
T -> REAL	T.tipo = real
L -> L_1, ID	L_1.her = L.her añadetipo(ID.ptr_tbs, L.her)
L -> ID	añadetipo(ID.ptr_tbs, L.her)

L.her = at. heredado

T.tipo = at. sintetizado

ID.ptr_tbs = ficticio (sintetizado)

GRAMÁTICA ATRIBUIDA COMPLETA

Una G.A. se dice completa si:

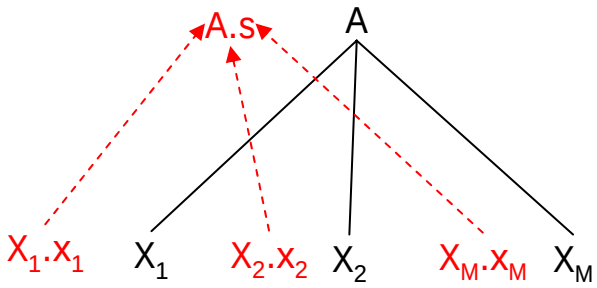
1. AS(X) U AH(X) = A(X)	Un atributo o es sintetizado o es heredado
2. AS(X) ∩ AH(X) = Ø	No puede ser a la vez sintetizado y heredado

3. Para todo X.a ∈ AS(X) y toda regla p ≡ X -> τ debe existir una regla semántica que calcule X.a, X.a=f(...) asociada a p

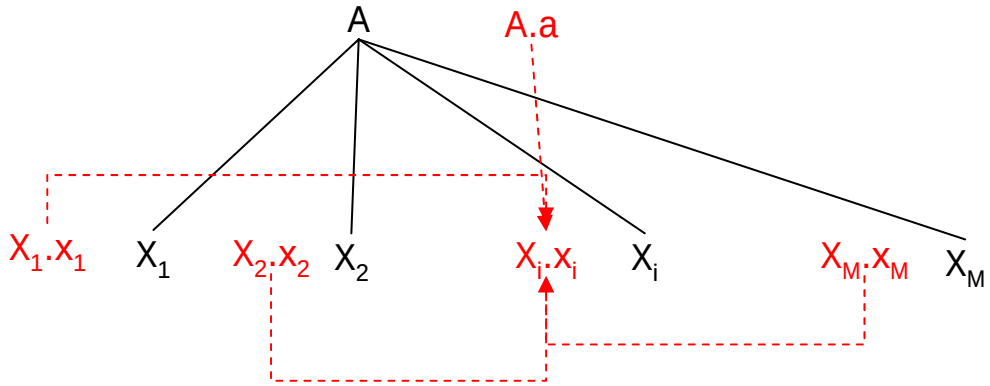
4. Para todo X.a ∈ AH(X) y toda regla p ≡ Y -> α X β debe existir una regla semántica que calcule X.a, X.a=f(...) asociada a p

Grafo de dependencias

- A -> $X_1X_2...X_M$ {A.s = f($X_1.x_1$, ..., $X_M.x_M$)}



- A -> $X_1X_2...X_M$ {$X_i.x_i$ = g($X_1.x_1$, ..., $X_M.x_M$, A.a)}



- Las **interdependencias** entre los atributos heredados y sintentizados en los nodos de un árbol sintáctico se representan mediante un **grafo de dependencias**.
- Conceptualmente, mediante una gramática atribuida, en primer lugar se construye el árbol sintáctico, luego el grafo de dependencias, y finalmente se evalúan las reglas semánticas.
- Un **ordenamiento topológico** de un grafo dirigido acíclico es todo ordenamiento m_1, ..., m_k de los nodos del grafo que respeta el sentido de las aristas m_i->m_j.

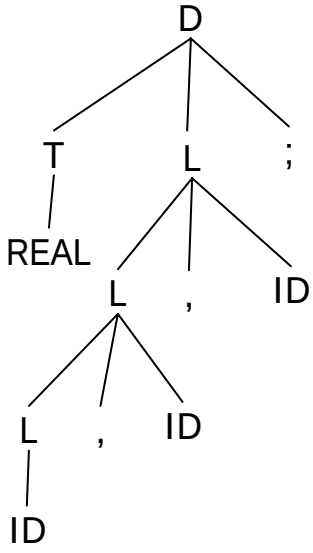
Gramática Atribuida para definición de tipos

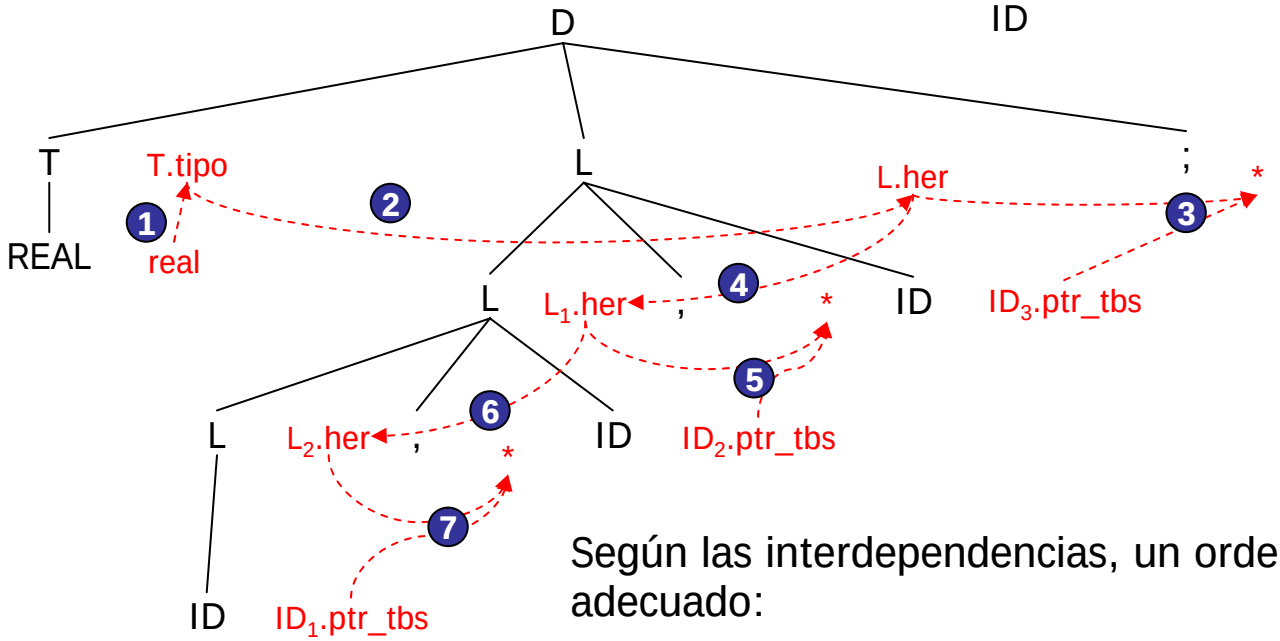
D -> T L ; L.her = T.tipo
T -> INT T.tipo = integer
T -> REAL T.tipo = real
L -> L_1, ID L_1.her = L.her
añadetipo(ID.ptr_tbs, L.her)
L -> ID añadetipo(ID.ptr_tbs, L.her)

real cont, valor, suma;

lexicográfico

REAL ID, ID, ID ;



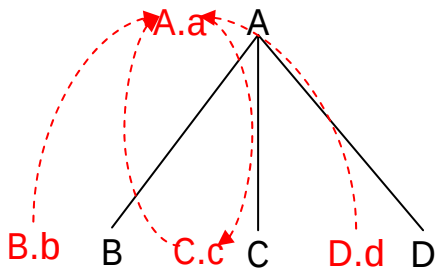


Según las interdependencias, un orden adecuado:

1) T.tipo=real
2) L.her=T.tipo
3) añade (ID_3.ptr_tbs, L.her)
4) L_1.her=L.her
5) añade (ID_2.ptr_tbs, L_1.her)
6) L_2.her=L_1.her
7) añade (ID_1.ptr_tbs, L_2.her)

Gramática L-atribuida

- Se dice que una gramática atribuida es **circular**, si el grafo de dependencias para un árbol sintáctico generado por la gramática tiene un ciclo.
- A -> B C D A.a = f(B.b, C.c, D.d), C.c = g(A.a)



- No se consigue encontrar un orden de evaluación sólo si el grado de dependencias tiene un ciclo.
- **GRAMÁTICA L-ATRIBUIDA** (ó atribuida por la izda. ó definición con atributos por la izda.)
  - En toda regla A -> $X_1X_2...X_M$, cada atributo heredado de Xj $1 \leq j \leq n$ depende sólo de:
    1. Los atributos (sintetizados o heredados) de los símbolos $X_1X_2...X_{j-1}$
    2. Los atributos heredados de A.
- Este tipo de gramáticas no produce ciclos y además admiten un **orden de evaluación en profundidad**

Visitaprof (nodo n)
{
  for cada hijo m de n, de izda a dcha
  {
    evaluar at. heredados de m;
    Visitaprof (m);
  }
  evaluar at. **sintetizados** de n;
}

Ejemplo de recorrido en profundidad

S -> C { C.t = 10; S.a = C.a; }

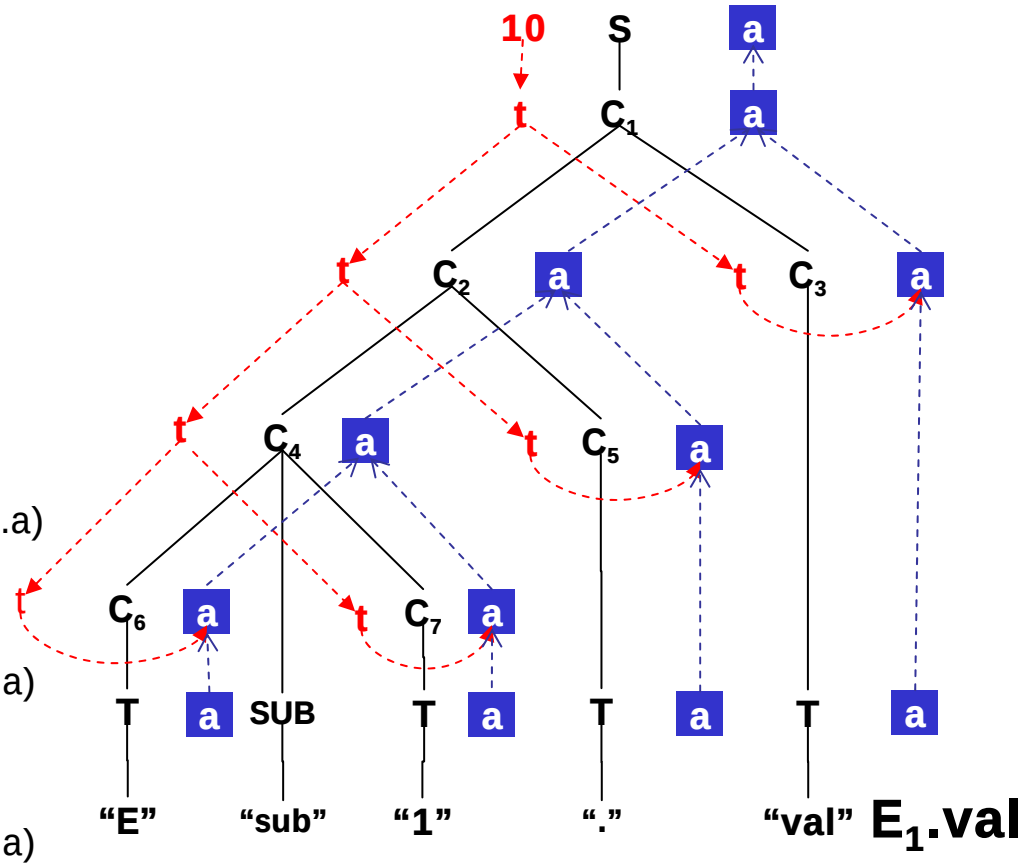
C -> C_1C_2 { C_1.t=C.t; C_2.t=C.t;
C.a=max(C_1.a,C_2.a);}

C -> C_1 sub C_2 {C_1.t=C.t; C_2.t=contrae(C.t);
C.a=desp(C_1.a, C_2.a);}

C-> texto {C.a=texto.a x C.t}

Tokens = {T ≡ texto, SUB ≡ sub }

C_1.t = 10
C_2.t=C_1.t
C_4.t=C_2.t
C_6.t=C_4.t
C_6.a=T_1.axC_6.t
C_7.t=contrae(C_4.t)
C_7.a=T_2.axC_7.t
C_4.a=desp(C_6.a,C_7.a)
C_5.t=C_2.t
C_5.a=T_3.axC_5.t
C_2.a=max(C_4.a,C_5.a)
C_3.t=C_1.t
C_3.a=T_4.axC_3.t
C_1.a=max(C_2.a,C_3.a)
S.a=C_1.a

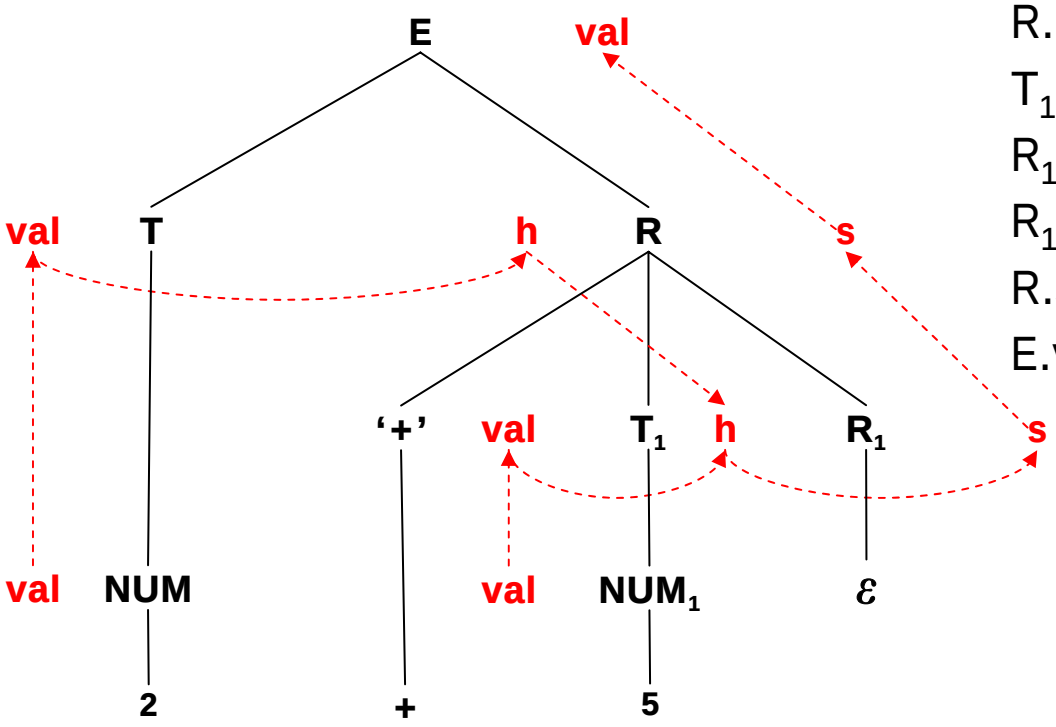


Recorrido LL(1) y evaluación en profundidad

- La gramática debe ser LL(1).
- Si la gramática es L-atribuida, se puede hacer entonces una evaluación en profundidad.
- Puede hacerse la evaluación mientras se realiza el análisis descendente LL(1)

E -> TR	{R.h= T.val; E.val=R.s;}
R -> +TR_1	{R_1.h=R.h+T.val; R.s=R_1.s;}
R -> -TR_1	{R_1.h=R.h-T.val; R.s=R_1.s;}
R -> ε	{R.s=R.h;}
T->(E)	{T.val=E.val;}
T->NUM	{T.val=NUM.val;}

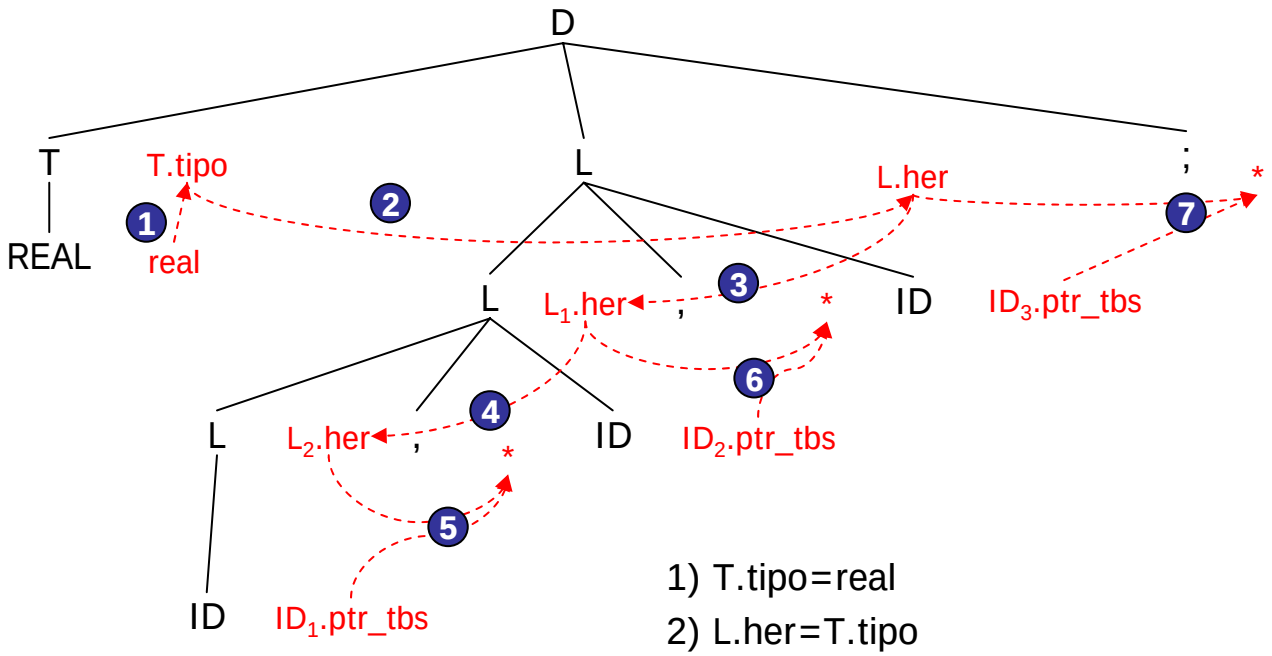
T.val=NUM.val
R.h=T.val
T_1.val=NUM_1.val
R_1.h=R.h+T_1.val
R_1.s=R_1.h
R.s=R_1.s
E.val=R.s



Gramática L-Atribuida pero no LL(1)

D -> T L ;	L.her = T.tipo
T -> INT	T.tipo = integer
T -> REAL	T.tipo = real
L -> L_1, ID	L_1.her = L.her
	*=añadetipo(ID.ptr_tbs, L.her)
L -> ID	*=añadetipo(ID.ptr_tbs, L.her)

- No se puede hacer simultáneamente el análisis LL(1) (no lo es) y el recorrido primero en profundidad.
- Si consideramos "*" como un atributo ficticio y sintetizado (asociado al padre), y hacemos un recorrido primero en profundidad en el grafo de interdependencias, la secuencia de evaluación de las reglas quedaría:



1) T.tipo=real
2) L.her=T.tipo
3) L_1.her=L.her
4) L_2.her=L_1.her
5) añade (ID_1.ptr_tbs, L_2.her)
6) añade (ID_2.ptr_tbs, L_1.her)
7) añade (ID_3.ptr_tbs, L.her)

Gramática S-atribuida

Es una gramática atribuida que sólo tiene atributos **sintetizados**

- Una gramática S-atribuida es también L-atribuida
- EVALUACIÓN **ASCENDENTE** de una gramática S-atribuida

L -> E '\n'	{print (E.val)}
E -> E_1+T	{E.val=E_1.val + T.val}
E -> T	{E.val=T.val}
T -> T_1*F	{T.val=T_1.val*F.val}
T -> F	{T.val=F.val}
F -> (E)	{F.val=E.val}
F -> DIGITO	{F.val=DIGITO.valex}

- **EJEMPLO**: 3 + 7 '\n'

ÁRBOL SINTÁCTICO	PILA ANÁLISIS	PILA ATRIBUTOS	REGLAS SEMA.
DIGITO 3	DIGITO	DIGITO.valex = 3	
F DIGITO 3	F	F.val = 3	F.val = D.valex $$ = $1
E, T, F, DIGITO(3), +, T, F, DIGITO(7)	T + E	T.val = 7 <-- $1 -- <-- $2 E.val = 3 <-- $3	
E, E, +, T, T, F, DIGITO(3), F, DIGITO(7)	E	E.val=10	E.val = E_1.val + T.val $$ = $1+$3

Esquema de Traducción (atribuido)

- Es una gramática atribuida en la cual las reglas semánticas aparecen intercaladas (entre { }) entre los símbolos del consecuente de una regla de producción, indicando así el momento en el que deben ser ejecutadas.
- Esquema de traducción atribuido **bien definido**, es aquél que debe cumplir:
  1. Un at. **heredado** asociado a un símbolo en el consecuente de una regla, debe ser evaluado **antes que** ese símbolo.
  2. Una regla semántica no debe hacer referencia a los at. **sintetizados** de un símbolo que esté a la **derecha** de dicha regla.
  3. Un at. **sintentizado** para un antecedente se debe calcular **al final** del consecuente.
- A partir de una gramática L-atribuida es posible construir un esquema equivalente bien definido.

- **EJEMPLO**

D -> T {L.her = T.tipo} L

T -> INT {T.tipo = integer }

T -> REAL {T.tipo = real }

L -> {L_1.her=L.her} L_1, ID {añade(ID.ptr_tbs, L.her)}

L -> ID {añade (ID.ptr_tbs, L.her)}

Ejemplo Esquema de Tradución bien definido

E -> T {R.her = T.val} R {E.val=R.s}

R -> + T {R_1.her=R.her+T.val} R_1 {R.s = R_1.s}

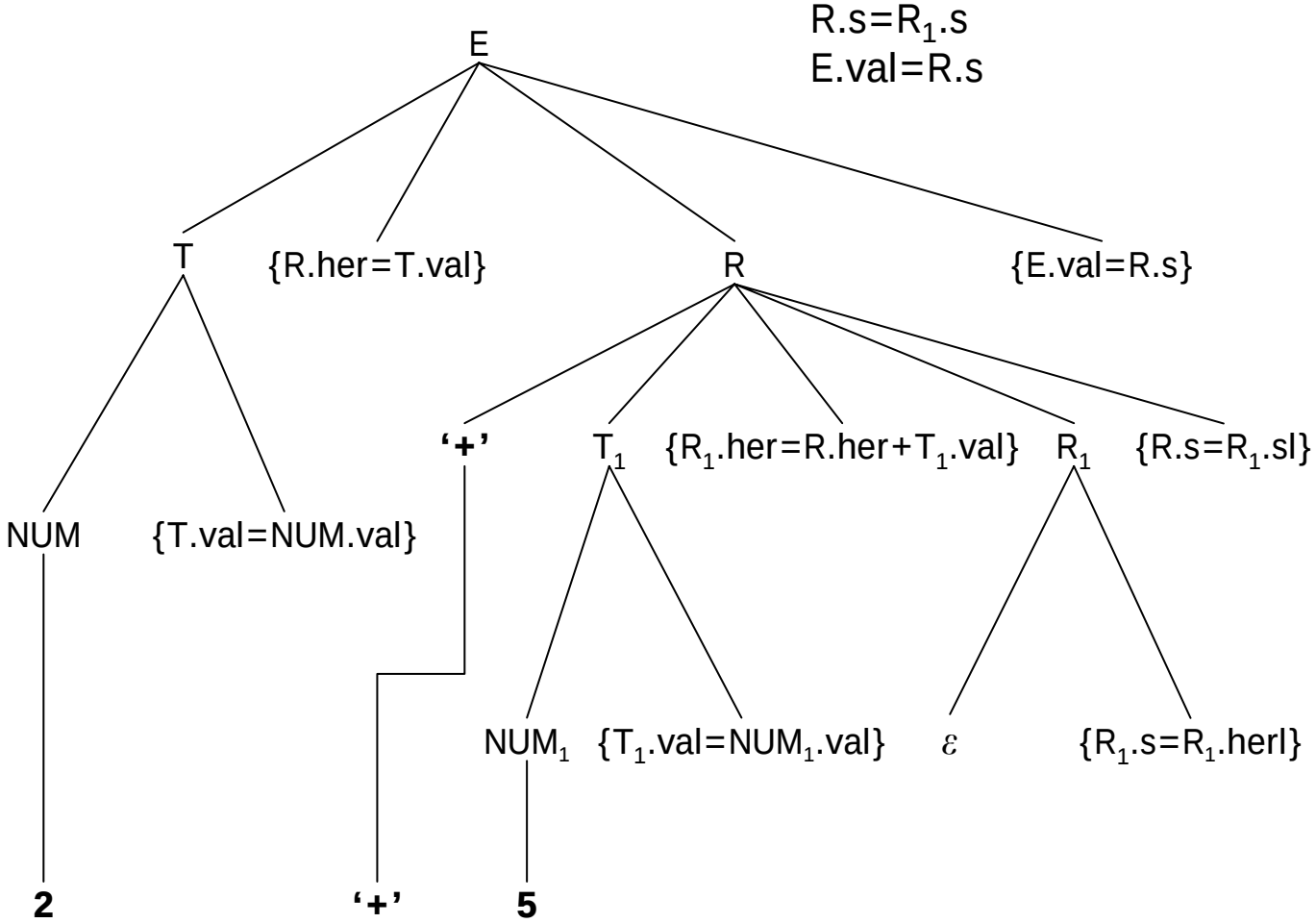
R -> - T {R_1.her = R.her – T.val} R_1 {R.s = R_1.s}

R -> ε {R.s=R.her}

T -> (E) {T.val=E.val}

T -> NUM {T.val = NUM.val}

T.val=NUM.val
R.her=T.val
T_1.val=NUM_1.val
R_1.her=R.her+T_1.val
R_1.s=R_1.her
R.s=R_1.s
E.val=R.s



Comprobación de tipos

- Comprobar los tipos mediante un esquema de traducción

Programa -> D ; E

D -> D ; D

D -> ID : T {añadetipo(ID.ptr_tbs, T.tipo}

T -> CHAR {T.tipo = **char** }

T -> INT {T.tipo = **int**}

T -> ^T_1 {T.tipo = **pointer** (T_1.tipo)}

T -> ARRAY [NUM] OF T_1
{T.tipo=**array**(1..NUM.val, T_1.tipo)}

...

E -> NUM {E.tipo = **int**}

E -> ID {E.tipo = busca (ID.ptr_tbs)}

E -> $E_1[E_2]$ {E.tipo = if E_2.tipo==**int** and
E_1.tipo=**array**(s,t) then t else error_tipo}

...

E -> REAL {E.tipo=**real**}

E ->E_1 OP E_2 {if E_1.tipo==**int** and E_2.tipo==**int**
then **int**
else if E_1.tipo==**int** and E_2.tipo==**real**
then **real**
else if E_1.tipo==**real** and E_2.tipo==**int**
then **real**
else if E_1.tipo==**real** and E_2.tipo==**real**
then **real**
else error_tipo }