Análisis Semántico

Función del analizador semántico.

Tabla de símbolos

- Función.
- Características.
- Contenido.
- Estructura.
- Operaciones.

Análisis dirigido por sintaxis.

Gramática con atributos.

- Métodos de evaluación de los atributos.
- Tipos de gramáticas con atributos.
- Esquemas de traducción.

Comprobación de tipos.

Conversión de tipos.

Bibliografía básica

[Aho90]	Alfred V. Aho, Ravi Sethi, Jffrey D. Ullman *Compiladores. Principios, técnicas y herramientas.* Addison-Wesley Iberoamericana 1990.
[Trem85]	J. Tremblay, P.G. Sorenson *The theory and practice of compiler writing.* Mc-Graw-Hill 1985.

TABLA DE SÍMBOLOS

Objetivo.

Delimitar el contexto de un conjunto de frases o instrucciones de un lenguaje libre de contexto.

Desde el punto del analizador sintáctico, lo único que se necesita identificar es que determinadas palabras (pertenecientes a un COMPONENTE SINTÁCTICO) deben aparecer precedidas o seguidas por otras palabras.

Utilidad.

Simplifican el análisis sintáctico.

Ayudan en la comprobaciones SEMÁNTICAS.

Ayudan en la generación de código.

Contenido de la tabla de símbolos.

Esencialmente la información que aparece en la tabla de símbolos es de dos tipos:

- El propio símbolo, y
- Los atributos necesarios para definir el símbolo a nivel semántico y de generación de código.

Los atributos requeridos para cada símbolo depende a nivel general si:

- del tipo de gestión de memoria,
- el lenguaje está, o no, estructurado en bloques,
- el símbolo es, o no, parámetros de un procedimiento o función.

CONSTRUCCIÓN DE LA TABLA DE SÍMBOLOS.

1. El analizador de léxico deberá:

    - Insertar los símbolo detectados en la tabla de símbolos,
    - Crear la tabla de símbolos parcialmente,
    - Señalar la línea del programa fuente en donde aparecen.

2. El analizador semántico:

    - Añadir los tipos, si procede, a los símbolos que aparecen en la tabla de símbolos.

Cuando se hace una implantación de una única pasada, entonces los símbolos son insertados y calificados, a nivel semántica y de generación de código, por el analizador sintáctico y en el mismo instante que son detectados por el analizador léxico y que son pasados al analizador sintáctico.

Operaciones sobre la tabla de símbolos.

- INSERTAR
- CONSULTAR
- MODIFICAR (añadir atributos nuevos)

El CUANDO y el CÓMO se usan estas operaciones dependen del tipo de lenguaje:

Lenguajes con DECLARACIONES DE VARIABLES:

- Explícitas:
    - Declaraciones: sólo INSERTAR.
    - Referencia: sólo CONSULTAR.
- Implícitas:
    - CONSULTAR si no está ya incluida.
    - INSERTAR, en caso contrario.
    - Lenguajes con estructura de BLOQUE : CREAR SUBTABLAS.

IMPLEMENTACIÓN DE LA TABLA DE SÍMBOLOS

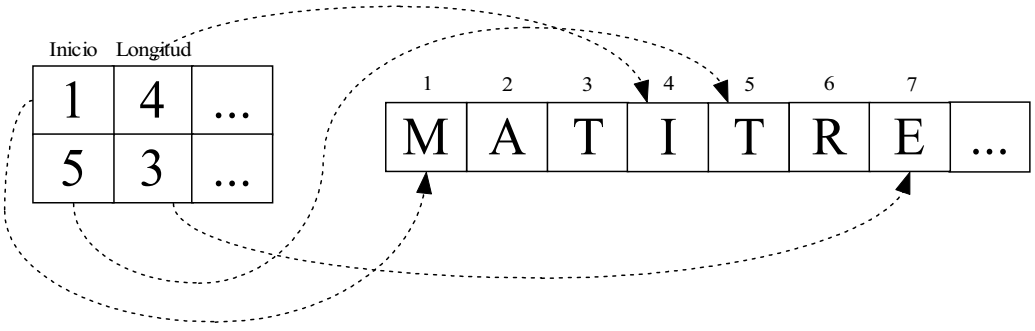
La distribución de la información de la tabla de símbolos dependerá de las características del lenguaje y de las restricciones establecidas para los símbolos.

Campo dedicado para el símbolo

- *Formato fijo*
  Apropiado cuando se establece límite en el número de caracteres que forman los símbolos y, además, sea pequeño.

  En este caso sólo se dispone de un área fija en la tabla para almacenar el símbolo.

- *Formato variable*
  Se dispone de la tabla de símbolos y de un área auxiliar en donde se introducen los símbolos de modo consecutivo. En la TS se sustituye el campo dedicado para el nombre del símbolo por un puntero al área auxiliar y un entero que indica la longitud del mismo.



Campo "dirección"

- *Lenguajes SIN estructura de Bloque.*
  Se asignan direcciones consecutivas según el orden en el que aparecen declaradas.

- *Lenguajes CON estructura de Bloque.*
  Para cada bloque se asigna una subtabla, la dirección será consecutiva para cada bloque

  Se necesitan dos campos:

Nº Bloque	Dirección Bloque

  Se introduce este campo en la TS cuando se declara. Se utiliza este atributo en la fase de Generación de código.

Campo "Tipo"

Se introduce cuando se identifica una declaración explícita o implícita de una variable.

Se utiliza para determinar la *memoria de almacenamiento* y para la *comprobación de tipos*.

Campo Nº de dimensiones / Nºde parámetros

La realiza el analizador semántico y sirve para delimitar el tamaño de memoria necesaria para representar el símbolo.
(Ejemplo: el tamaño o dimensión de un array; si se trata de una función o procedimiento, el número de argumentos que posee y sus tipos para la reserva de memoria).

Campos Lista cruzada de referencia y Puntero de orden

Son útiles para el programador del traductor con objeto de facilitar el uso de la TS.

ORGANIZACIÓN DE LA TABLA DE SÍMBOLOS

a) **Lenguajes SIN estructura de Bloque**

- LISTA
  - Ordenada
    - Árboles binarios
    - Tablas Hash (usando encadenamiento)
  - No ordenada

b) **Lenguajes CON estructura de Bloque**

- Una tabla para cada Bloque con estructura de PILA.
- Estructura ➔ Consta de una PILA + Índice (que apunta al inicio de las variables pertenecientes a cada bloque).

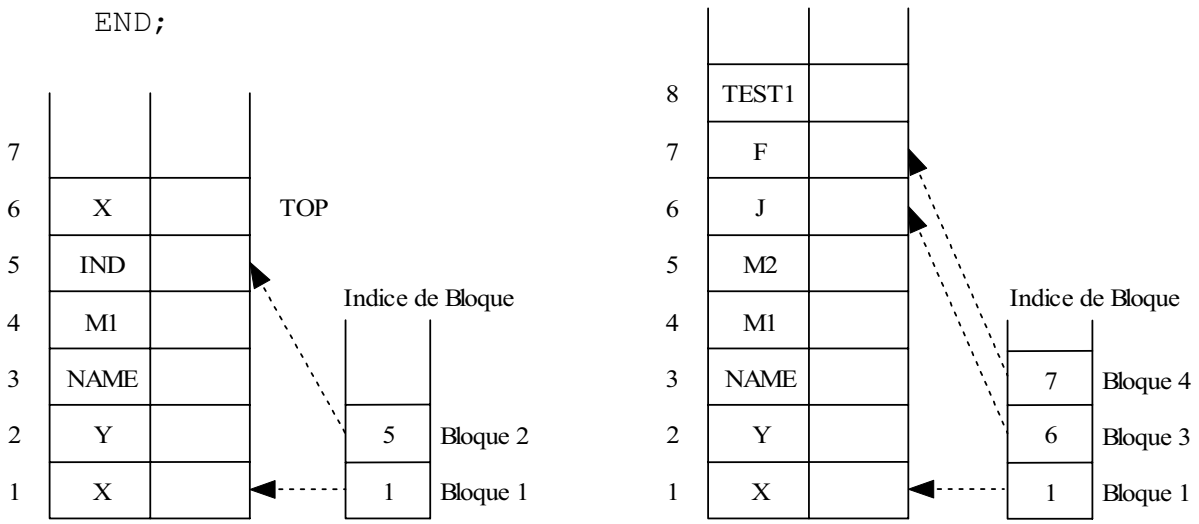
El índice nos muestra el comienzo y final de un bloque.

Además de insertar, consultar y modificar (completar la información de un símbolo), cuando se trabaja con bloques, se realizan además las operaciones siguientes:

- *Set* (Marca de Inicio de bloque).
- *Reset* (Marca de Fin de bloque).

Ejemplo 1. Dada la secuencia de instrucciones siguientes. Se construye la tabla de símbolos de la siguiente forma:

```
Block
     Real x, y;
     String NAME;

     M1 : PBLOCK (Integer IND) ;
                  Integer x;
                  ...
                  CALL M2 (IND+1) ;
                  ...
          END M1 ;

     M2 : PBLOCK (Integer J) ;
                  PBLOCK
                       Array Integer (F(J));
                       Logical TEST ;
                       ...
                  END ;
     END M2 ;
     ...
     CALL M(X/Y) ;
     ...
END;
```



Operación	Símbolos activos	Símbolos inactivos
Set BLK1		
Set BLK2	M1, NAME, Y, X	
Reset BLK2	X, IND, M1, NAME, Y, X	
Set BLK3	M2, M1, NAME, Y, X	X, IND
Set BLK4	J, M2, M1, NAME, Y, X	X, IND
Reset BLK4	TEST, F, J, M2, M1, NAME, Y,X	X, IND
Reset BLK3	J, M2, M1, NAME, Y, X	TEST, F, X, IND
Reset BLK1	M2, M1, NAME, Y, X	J, TEST, F, X, IND
Fin Compilación		M2, M1, NAME, Y, X, J, TEST, F, X, IND

GRAMÁTICA CON ATRIBUTOS

Una gramática con atributos es una gramática de contexto libre cuyos símbolos pueden tener asociados atributos y las producciones pueden tener asociadas reglas de evaluación de los atributos.

Cada símbolo puede tener asociado un número finito de atributos.
Cada producción puede tener asociada un número finito de reglas de evaluación de los atributos.
Los valores de los atributos deberán estar asociados con un dominio de valores.

Sea $G = (T, N, P, S)$ una gramática de contexto libre y $C = \{c_1, c_2, \ldots, c_n\}$ el conjunto de atributos asociados con los símbolos de la gramática G.

Dada una regla de evaluación $b = f(c_1, \ldots, c_k)$ asociado con la producción $A \rightarrow \alpha \in P$.

Atributo Heredado: Si b está asociado con algún símbolo de α.

Atributo Sintetizado: Si b está asociado con el símbolo no terminal A.

ATRIBUTOS SINTETIZADOS

Evaluación:

Las reglas de evaluación de los atributos sintetizados se realizan cuando se aplican reducciones en el análisis sintáctico.

Requisitos para la evaluación:

. Las reglas de evaluación de los atributos deben definirse en función de los atributos asociados con los símbolos gramaticales a su derecha.

. Realizar un análisis ascendente.

Gramáticas S-Atribuidas:

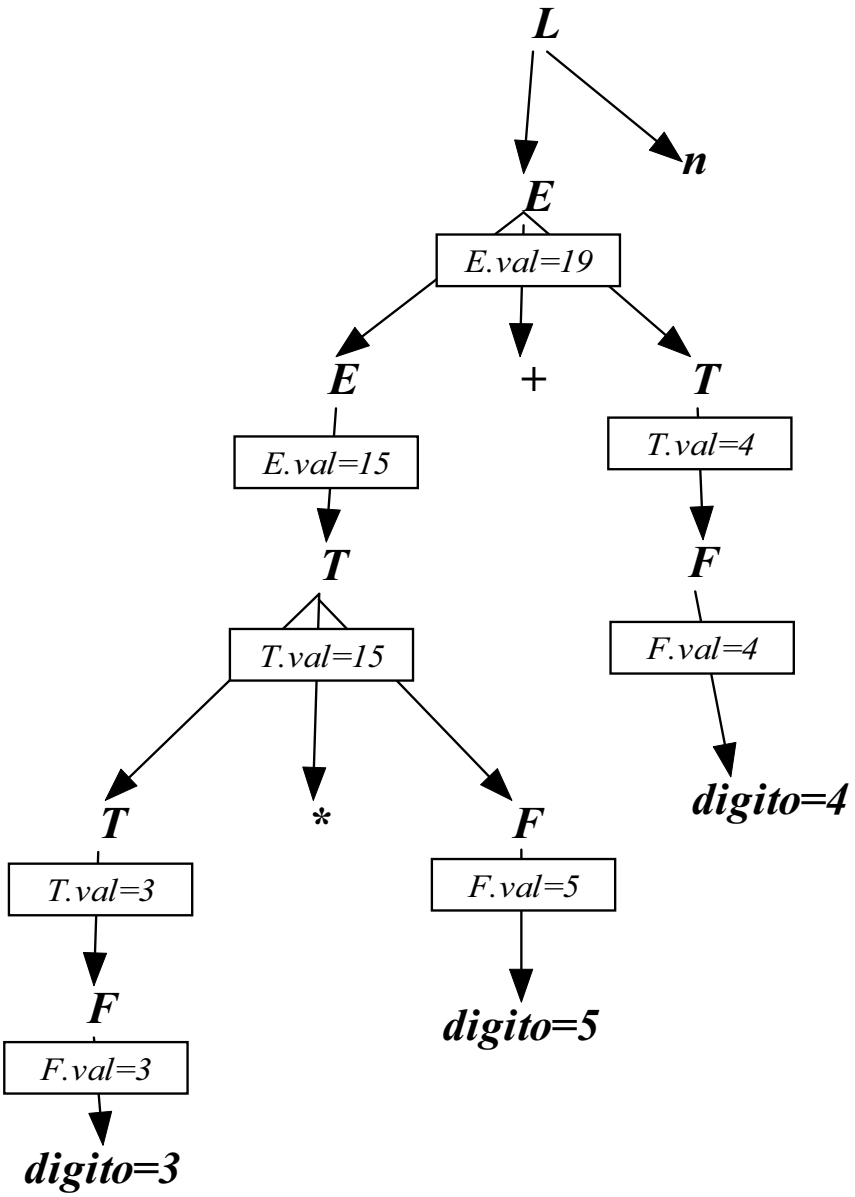
Cuando todos los atributos asociados con los símbolos gramaticales son sintetizados.

Ejemplo 2. Se muestra un ejemplo de gramática S-Atribuida:
Deseamos evaluar expresiones a la vez que la analizamos:

Sea el conjunto de producciones y acciones siguientes:

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

La evaluacion de la expresión: `3*5+4n`



ATRIBUTOS HEREDADOS

La evaluación de un atributo heredado depende de los atributos asociados con los símbolos precedentes en la derivación.

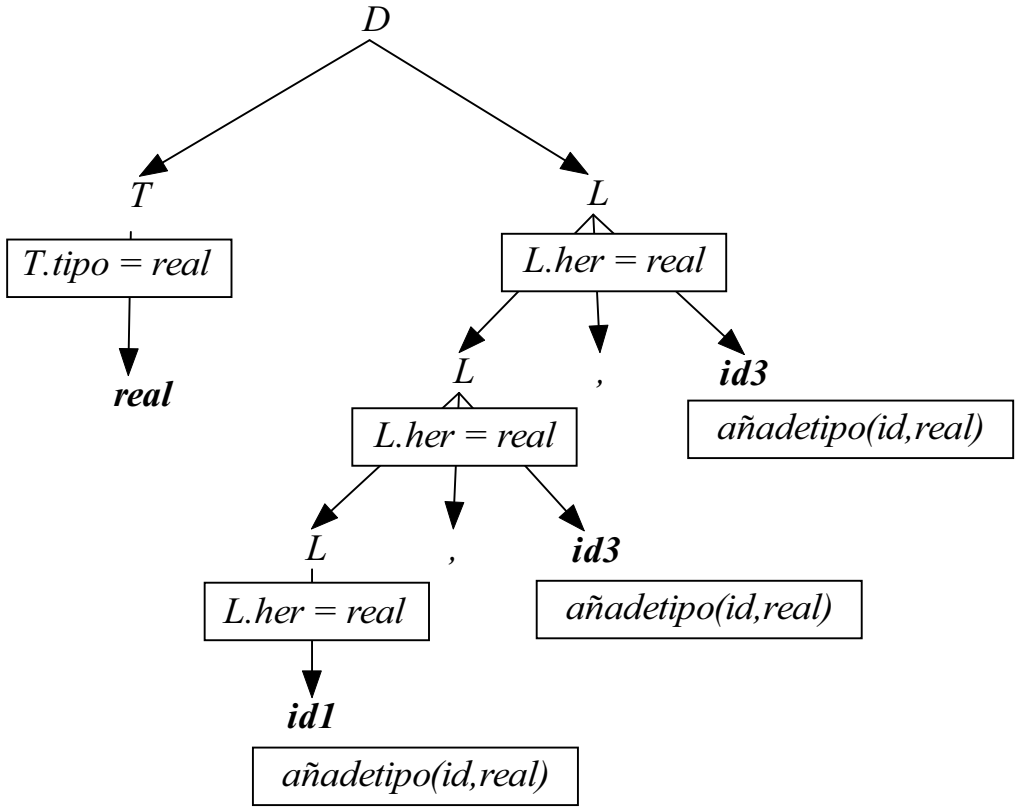
Requisitos:

Realizar un análisis descendente.

Dada la gramática y las reglas de evaluación siguientes:

Producciones	Reglas
$D \rightarrow T\ L$	$L_2.in = T_1.tipo;$
$T \rightarrow$ int	$T_0.tipo = entero;$
$T \rightarrow$ real	$T_0.tipo = real;$
$L \rightarrow L$, id	$L_1.in = L_0.in;$
	$añadetipo(id, L_0.in);$
$L \rightarrow$ id	$añadetipo(id, L_0.in);$

```
Real id1,id2,id3
```



GRAFOS DE DEPENDENCIA.

Cuando aparecen definidos atributos sintetizados y heredados, es necesario establecer un ORDEN DE EVALUACIÓN DE LOS ATRIBUTOS

El grafo de dependencia es un grafo dirigido construido por el algoritmo siguiente:

Nodos: Para cada nodo del árbol de análisis ***n*** *hacer:*
Para cada atributo ***s*** *asociado a Los símbolos del nodo* ***n*** *hacer:*
Construir un nodo etiquetado con ***a***.

Arcos: Para cada regla semántica $\boldsymbol{b=f(e_1,...,e_n)}$ *asociada con la producción del nodo* ***n*** *hacer:*
Para $\boldsymbol{i=1,..n}$ *Hacer: Trazar arcos desde* $\boldsymbol{c_i}$ *a* ***b***.

Métodos de Evaluación de las Reglas Semánticas:

1. Árbol de análisis: *para cada entrada se construye el árbol sintáctico* **(Grafos cíclicos)**.
2. Basado en las reglas semánticas: *Dependiendo de las reglas semánticas (atributos heredados o sintetizados) se establece el orden de evaluación.*
3. Dirigido por sintaxis: *El orden de evaluación es impuesto por la estrategia de análisis.*

EVALUACIÓN ASCENDENTE

Gramáticas S-Atribuidas:

La estructura de la pila se adecua para que cada símbolo de la gramática disponga de sus atributos asociados.

La evaluación de los atributos se realiza cuando se REDUCE.

0	...		...
5	X		X.x, X.y,...X.z
6	Y		Y.u,...,Y.s
7	Z		Z.a,...,Z.c
	...		...

Gramáticas L-Atribuidas:
Sea la producción $A \rightarrow X_1 X_2 ... X_n$. Una gramática es L-atribuida si todos los atributos Heredados asociados con X_j, $1 \leq j \leq n$, sólo depende:

1. de los atributos asociados con los símbolos $X_1 X_2 ... X_{j-1}$
2. de atributos asociados con A

TODA GRAMÁTICA L-ATRIBUIDA PUEDE SER EVALUADA MEDIANTE ANÁLISIS ASCENDENTE.

COMPROBACIONES SEMÁNTICAS

Comprobaciones ESTÁTICAS.

Las comprobaciones sintácticas y semánticas.

Comprobaciones DINÁMICAS.

Realizadas en tiempo de ejecución.

Comprobaciones SEMÁNTICAS

De TIPO.
Verificación del tipo de los operandos en las expresiones.

De FLUJO de CONTROL.
Verifica los puntos del programa de salida y entrada del control.

De UNICIDAD.
Verifica la presencia de símbolos de forma única. (ejemplo: declarar un símbolo una sóla vez).

Relación de NOMBRES.
Un mismo nombre puede aparecer más de una vez.

EXPRESIONES DE TIPOS

- **Tipos básicos** (Tipos simples) son expresiones de tipo.

  Ejemplo: boolean, char, integer, real, ...

- Una expresión de tipos puede ser nombrada por un **nombre**. Un nombre de tipos es una expresión de tipos.

En una expresión de tipo pueden aparecer VARIABLES DE TIPOS.

CONSTRUCTORES DE TIPOS

- **Matrices** (Arrays)
  Array (I, T) : Expresión de tipo que representa: Tipo de una matriz con elementos de tipo T y conjunto de índices I.

  En Pascal:

```
VAR A: array [1..10] of integer;
```

  ⇓

  Le asocia la expresión tipo: array (1..10, integer) a A

- **Producto**
  Sea T_1 y T_2 expresiones de tipo, entonces $T_1 \times T_2$ (producto cartesiano) es también una expresión de tipo.

- **Registro**
  Es como producto pero con nombre.

  Record (dirección×integer) ×(lexema×array(1..15,char)))

```
type file=record
            direccion : integer ;
            lexema : array[1..15] of char ;
          end ;
```

- **Apuntadores**

  Sea *T* una expresión de tipo, entonces *pointer*(*T*) es una expresión de tipo.

  Pointer(FILA) : apunta a un objeto tipo FILA

  ⇓

```
VAR p : ^FILA
```

- **Funciones**

  Como una función matemática

  Sean *D* y *R* dos expresiones de tipos entonces:

  $D \rightarrow R$, es otra expresión de tipo

  Ejemplo:

  char×char → *pointer*(integer)

  ⇓

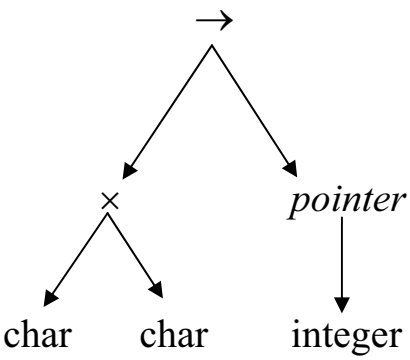
```
function f(a, b : char) : ^integer;
```

REPRESENTACIÓN DE LAS EXPRESIONES DE TIPOS

Mediante grafo dirigido acíclico (GDA)

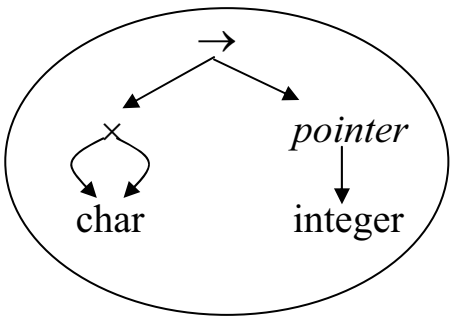
Ejemplo:

char×char → *pointer*(integer)



hojas interiores: constructor de expresiones o expresiones de tipos.

hojas finales: tipos básicos.



Representación de expresiones de tipos por mapa de bit
(*usado por D.M. Ritchie para el compilador de C*)

Se parte de los constructores básicos:

	Codificación
pointer (*T*)	: Expresión Puntero → 01
freturn (*T*)	: Expresión función → 11
array (*T*)	: Expresión array → 10

Tipos básicos	Codificación
Boolean	0000
Char	0001
Integer	0010
Real	0011

Representación de las expresiones de tipo

Expresiones de Tipo	Mapa de Bit
char	000000 0001
freturn (*char*)	000011 0001
pointer (*freturn* (*char*))	000111 0001
array (*pointer* (*freturn* (*char*)))	100111 0001

La principal **ventaja** es que se ahorra espacio:

Dos secuencias de bit distintas representan expresiones de tipos distintos.

Pero dos secuencias de bit iguales no podemos asegurar que se correspondan con tipos equivalentes.

Equivalencia estructural

Dos expresiones de tipos son equivalentes estructuralmente si responde a un mismo tipo básico o está formada a base de aplicar el mismo constructor de tipos sobre expresiones equivalentes estructuralmente o los mismos tipos básicos.

Modo de determinar si dos expresiones de tipos son equivalentes:

a) Aplicando GDA.
b) Algoritmo de análisis de los constructores.

Algoritmo de equivalencia estructural

function *equivest* (*s*, *t*) : **boolean** ;
begin
    if *s* y *t* son el mismo *tipo_básico* **then**
        return true ;
    else if $s = array(s_2,s_2)$ **and** $t = array(t_1,t_2)$ **then**
        return *equivest* (s_1,t_1) **and** *equivest* (s_2,t_2);
    else if $s = s_1 \times s_2$ and $t = t_1 \times t_2$ **then**
        return *equivest* (s_1,t_1) **and** *equivest* (s_2,t_2);
    else if $s = pointer(s_1)$ **and** $t = pointer(t_1)$ **then**
        return *equivest* (s_1,t_1);
    else if $s = s_1 \rightarrow s_2$ **and** $t_1 = t_1 \rightarrow t_2$ **then**
        return *equivest* (s_1,t_1) **and** *equivest* (s_2,t_2)
    else
        return false ;
end;

Equivalencia de nombre

Cuando las expresiones de tipos son nombradas, entonces dos expresiones de tipos son **equivalentes de nombre** si y solo si ambas expresiones son idénticas.

Si aparecen expresiones de tipos nombradas, también ha de ser reconsiderado el concepto de *equivalencia estructural*.

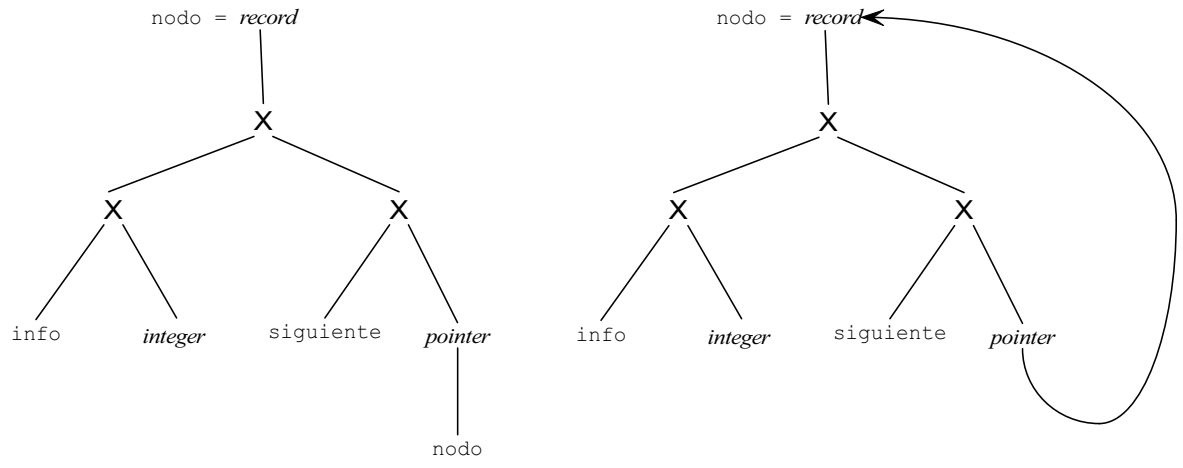
Dos expresiones de tipos son *equivalentes estructuralmente* si sustituídos los nombres de tipos por sus correspondientes expresiones de tipos, resultan dos nuevas expresiones de tipos que son equivalentes estructuralmente.

Ejemplo 3. Definición de registros en C y PASCAL.

```
/*  Lenguaje C */

struct nodo {
    int info ;
    struct nodo *siguiente ;
} ;
```

```
{ Lenguaje PASCAL }

type enlace = ^nodo ;
nodo = RECORD
         info : integer ;
         siguiente : enlace ;
END ;
```

La declaración en C evita los ciclos en los grafos de tipos utilizando la equivalencia estructural para todos los tipos excepto en los registros.



(a) (b)

PROBLEMA: estudio de la equivalencia en grafos cíclicos.

ESPECIFICACIÓN DE UN COMPROBADOR DE TIPOS ELEMENTAL

Un comprobador de tipos ha de disponer de:

- Asignación de tipos.
- Comprobador de tipos en las expresiones.
- Comprobador de tipos en las proposiciones o sentencias.
- Comprobador de tipos de las funciones.

Básicamente se advierten dos tareas:

- De asignación.
- De evaluación y comprobación.

Ejemplo 4. Sea un lenguaje sencillo que obedece a una gramática con el conjunto de producciones siguientes:

$P \rightarrow D$ **;** E **;** S
$D \rightarrow D$ **;** D
$D \rightarrow id$ **:** $T \mid \varepsilon$
$T \rightarrow$ **char** | **integer** | **array [**num**] of** $T \mid$ **^**$T \mid T$ **->** T
$E \rightarrow$ literal | num | id | E mod $E \mid E[E] \mid E$**^** $\mid E$**(**E**)** $\mid \varepsilon$
$S \rightarrow$ id **:=** $E \mid$ **if** E **then** $S \mid$ **while** E **do** $S \mid S$ **;** S

a) Asignación de tipos.
b) Comprobación de tipos en expresiones.
c) Comprobación de tipos en proposiciones.
d) Comprobación de tipos en funciones.

a) Asignación de tipos

$P \rightarrow D$ **;** E **;** S

$D \rightarrow D$; D

$D \rightarrow$ id **:** T { *añadetipo*(id.entrada, T.tipo); }

$T \rightarrow$ **char** { T.tipo := char ; }

$T \rightarrow$ **integer** { T.tipo := integer ; }

$T \rightarrow$ ^T_1 { T.tipo := *pointer* (T.tipo) ; }

$T \rightarrow$ **array[**num**] of** T_1 { T.tipo := *array* (1..num.val, T_1.tipo) ; }

$T \rightarrow T_1$ -> T_2 { T.tipo := T_1.tipo $\rightarrow$ T_2.tipo; }

b) Comprobación de tipos en las expresiones

$E \rightarrow$ literal { E.tipo := char ; }

$E \rightarrow$ num { E.tipo := integer ; }

$E \rightarrow$ id { E.tipo := *busca* (id.entrada) ; }

$E \rightarrow E_1$ mod E_2 { E.tipo := if E_1.tipo=integer and E_2.tipo=integer
then integer
else error_tipo; }

$E \rightarrow E_1[E_2]$ {E.tipo := if E_2.tipo=integer and E_1.tipo=*array* (s, t)
then t
else error_tipo; }

$E \rightarrow E_1$^ {E.tipo := if E_2.tipo=*pointer* (t)
then t
else error_tipo ; }

c) Comprobación del tipo en proposiciones

$S \rightarrow$ id **:=** E { id.tipo := *busca* (id.entrada);
S.tipo := if id.tipo=E.tipo
then vacio
else error_tipo; }

$S \rightarrow$ **if** E **then** S_1 { S.tipo := if E.tipo=boolean
then S_1.tipo
else error_tipo; }

$S \rightarrow$ **while** E **do** S_1 { S.tipo := if E.tipo=boolean
then S_1.tipo
else error_tipo; }

$S \rightarrow S_1$ **;** S_2 { S.tipo := if S_1.tipo=vacio and S_2.tipo=vacio
then vacio
else error_tipo; }

d) Comprobación del tipo en funciones

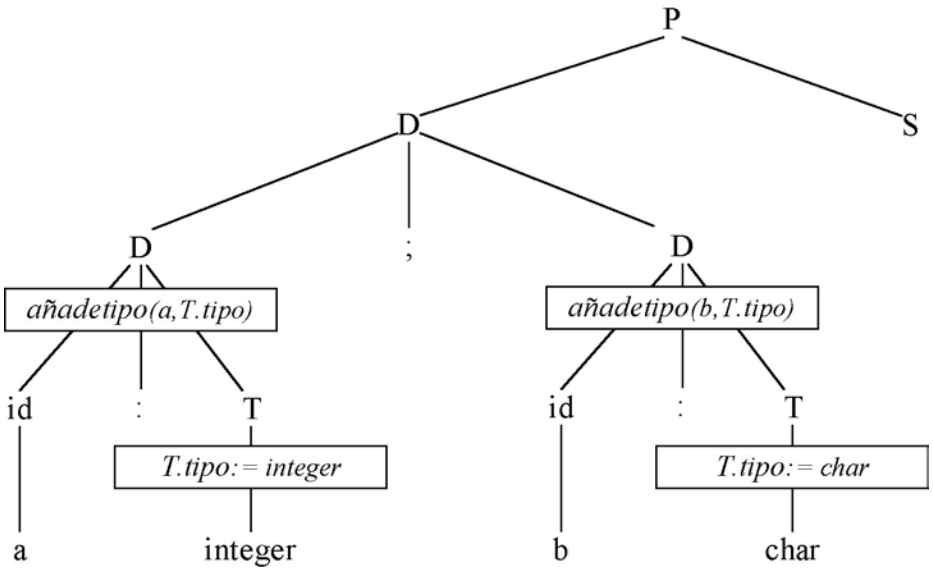
En la fase de declaraciones ha de ser definido el tipo

$E \rightarrow E_1(E_2)$ { E.tipo := if E_2.tipo= s and E_1.tipo= $s \rightarrow t$
then t
else error_tipo; }

Ejemplo 5. Dadas las sentencias válidas para la gramática definida en el ejemplo 4.

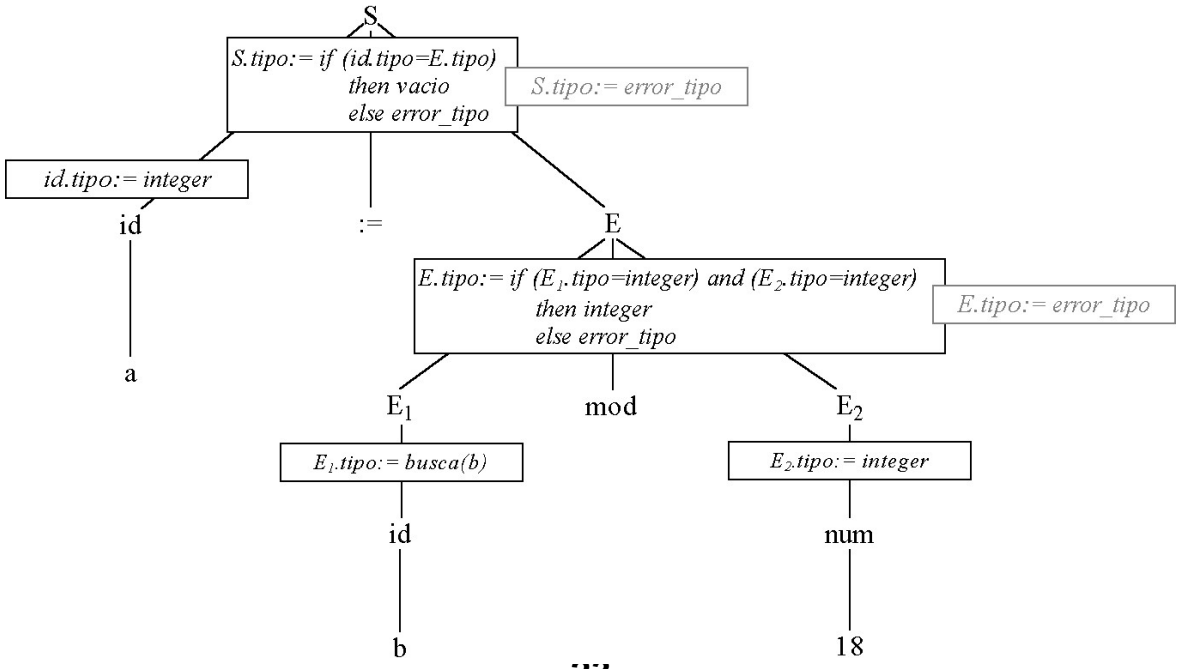
```
a : integer ;
b : char ;

a := b mod 18 ;
```

Tabla de símbolos	
...	...
a	integer
b	char
...	...



El árbol sintáctico para las declaraciones es:

De igual modo, el árbol sintáctico para la asignación es:



TÓPICOS EN LA COMPROBACIÓN DE TIPOS

- **Conversión de tipos (Coerción).**

Implícita:
Se realiza automáticamente por medio del compilador.

Explícita:
Se explicita por parte del programador la conversión.

Ejemplo 6. A partir de una gramática que admite tipos real y entero, se muestra la conversión de ambos ante una operación binaria.

$E \rightarrow$ ConstEntera { E.tipo := integer ; }

$E \rightarrow$ ConstReal { E.tipo := real ; }

$E \rightarrow$ id { E.tipo := *busca* (id.entrada) ; }

$E \rightarrow E_1$ **op** E_2 { E.tipo := if E_1.tipo=integer and E_2.tipo=integer
then integer
else if E_1.tipo=integer and E_2.tipo=real
then real
else if E_1.tipo=real and E_2.tipo=integer
then real
else if E_1.tipo=real and E_2.tipo=real
then real
else error_tipo ; }

- **Sobrecarga de Funciones y Operadores.**

+, -, son símbolos sobrecargados ya que su significado dependerá del contexto.

(,), pueden estar sobrecargados (ADA, FORTRAN), ya que se usan como referencia de elementos de matrices y como funciones. La sobrecarga se resuelve, en algunos casos, en base a los tipos de las expresiones en donde aparecen.