Procesadores de lenguaje

Tema 4 – Análisis semántico

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

Resumen

- Introducción
- Gramáticas de atributos.
  - Gramáticas S-atribuidas.
  - Gramáticas L-atribuidas.
- Esquemas de traducción dirigidos por sintaxis.
- Grafo de dependencias.
- Evaluación de atributos.

Introducción

- El lenguaje es un vehículo por el cual se transmiten instrucciones a un procesador para que las ejecute y produzca ciertos resultados.
- Es tarea del compilador extraer el contenido semántico incluido en las sentencias del programa.
- Ciertos aspectos relativos a la corrección de un programa no se pueden expresar claramente mediante el lenguaje de programación.
- Es necesario dotar al compilador de rutinas auxiliares para captar todo lo que no se ha expresado mediante la sintaxis del lenguaje

Introducción

- **Semántica:** ***conjunto de reglas que especifican el significado de cualquier sentencia sintácticamente correcta y escrita en un determinado lenguaje*.**
- El análisis semántico, a diferencia de otras fases, no se realiza claramente diferenciado del resto de las tareas del compilador.
  - Fase en la que se obtiene información necesaria para la compilación tras conocer la estructura sintáctica del programa.
  - Completa las fases de análisis léxico y sintáctico incorporando comprobaciones que no pueden asimilarse al mero reconocimiento de una cadena dentro de un lenguaje

Introducción

- Errores semánticos de un programa:
  - Conversiones de tipos no permitidas

```
int x;
x = 4.32;
Error: Ej1.java [6:1] possible loss of precision
```

  - Variables usadas y no definidas
  - Operandos de tipos no compatibles

```
if (x || 5) x = 0;
Error: Ej2.java [7:1] operator || cannot be applied to int,int
```


Funciones del análisis semántico

- Las principales funciones son:
  - Identificar cada tipo de instrucción y sus componentes.
  - Completar la ***Tabla de Símbolos***.
  - Realizar comprobaciones *estáticas*:
    - Se realizan durante la compilación del programa.
    - Ejemplos: comp. de tipos, unicidad de etiquetas e identificadores, etc.
  - Realizar comprobaciones *dinámicas*:
    - Aquellas que el compilador incorpora al programa traducido.
    - Hacen referencia a aspectos que sólo pueden ser conocidos en *tiempo de ejecución*
    - Dependientes del estado de la máquina en la ejecución o del propio programa.
  - Validar las declaraciones de identificadores: en muchos lenguajes no se puede usar una variable si no ha sido declarada con anterioridad.

Introducción

- El análisis semántico se divide en dos categorías:
  - Análisis de la exactitud del programa para garantizar una ejecución adecuada.
    - Algunos lenguajes (Lisp, Smalltallk) pueden no tener análisis estático.
    - Por ejemplo, ADA es un lenguaje con fuertes restricciones para que un programa sea ejecutable.
  - Análisis para mejorar la eficiencia (optimización del programa traducido)

Especificación de la semántica

- No hay una notación estándar para especificar la semántica estática de un lenguaje
  - El análisis semántico varía mucho de unos lenguajes a otros
- Las especificaciones semánticas de un lenguaje pueden hacerse de manera informal o formal:
  - Especificación natural: basada en el lenguaje natural.
    - Por ejemplo:
      - "Los identificadores deben definirse antes de utilizarse"
      - "Los operandos deben ser compatibles entre sí"
  - Especificación formal: definición más precisa.
    - Lenguajes formales: Z, B, VDM, etc.
    - Gramáticas de atributos (Knuth, 1968)

Gramáticas de atributos

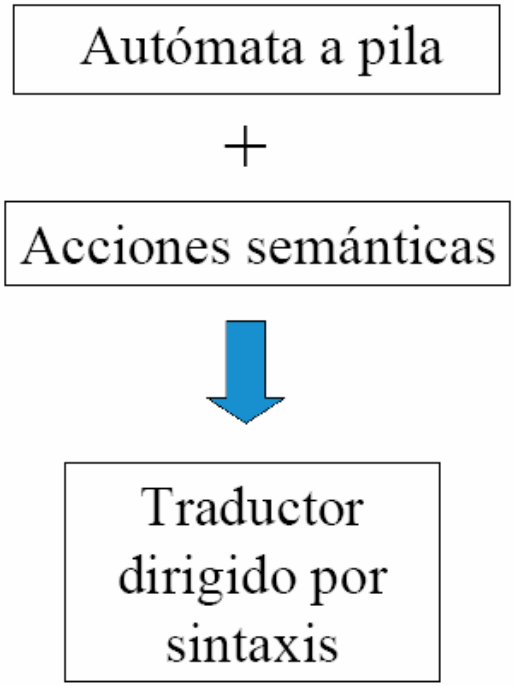
- Una gramática de atributos es una gramática libre de contexto cuyos símbolos pueden tener asociados atributos y las producciones pueden tener asociadas reglas de evaluación de los atributos.

- En la creación de compiladores se utilizan ecuaciones de atributos o reglas semánticas como método para expresar la relación entre el cálculo de los atributos y las reglas del lenguaje.

- Cada producción (regla sintáctica) tiene asociada una acción semántica que se aplica cuando se realiza una reducción en el análisis sintáctico ascendente.

Gramáticas de atributos

- Traducción dirigida por la sintaxis:



Gramáticas de atributos

- Dos notaciones para asociar reglas semánticas con producciones:
  - **Definiciones dirigidas por la sintaxis** (DDS) :
    - Son especificaciones de alto nivel
    - El usuario no necesita especificar el orden de la traducción
  - **Esquemas de traducción** (EDT) :
    - Indican el orden en que deben evaluarse las reglas semánticas
    - Incluyen detalles de implementación
- Con ambas notaciones se analizan los componentes léxicos, se construye el árbol sintáctico y finalmente se recorre el árbol para evaluar las reglas semánticas de sus nodos.

Gramáticas de atributos

- Atributo: propiedad de una construcción de un lenguaje.
  - Pueden variar mucho en cuanto a información que contienen o tiempo que tardan en determinarse durante la traducción/ejecución.
  - Cada símbolo (terminal o no terminal) puede tener asociado un número finito de atributos.

- Ejemplos de atributos:
  - Tipo de una variable
  - Valor de una expresión
  - Ubicación en memoria de una variable
  - Código objeto de un procedimiento
  - Número de dígitos significativos en un número

Gramáticas de atributos

- Fijación de un atributo: proceso de calcular el valor de un atributo y asociarlo con una construcción del lenguaje.

- Tipos de Atributo por su fijación:
  - Estático: puede fijarse antes de la ejecución del programa
    - Ej.: número de dígitos significativos (puede tener un valor mínimo)
  - Dinámico: sólo puede fijarse durante la ejecución del programa
    - Ej.: valor de una expresión no constante

- Los valores de los atributos deben estar asociados con un dominio de valores.

Gramáticas de atributos

- Generalmente se denotan mediante un nombre precedido por un punto y el nombre del símbolo al que están asociados.

NombreSímbolo.NombreAtributo

- Ejemplo:

```
numero→ numero digito | digito

a) numero→ digito
    numero.valor = digito.valor

b) numero→ numero digito
    numero1.valor = numero2.valor * 10 + digito.valor
```


Gramáticas de atributos

- Otra notación hace referencia a su posición en la regla de producción:
  - Se utiliza el símbolo '$'.
  - $$ representa el no terminal en la parte izquierda de la producción
  - Los símbolos de la parte derecha de la producción se identifican consecutivamente: $1, $2, $3, ..., $n.

- Ejemplo:

```
numero→ numero digito | digito
a) numero → digito
     $$.valor = $1.valor
b) numero → numero digito
     $$.valor = $1.valor * 10 + $2.valor
```


Gramáticas de atributos

- Ejemplo:

```
Exp → Exp op_arit Exp   {
   si  ($1.tipo == $3.tipo) entonces
        $$.tipo = $1.tipo
   si no
        $$.tipo = ERROR
        Escribir("error tipos incompatibles")
  fin_si
}
```


Gramáticas de atributos

```
<Expresion> ::= <Expresion> <Operador> <Expresion> {
    <Operador>.Tipo = Mayor_tipo(<expresion2>.Tipo,<expresion3>.Tipo)
    <Expresion1>.Tipo = <Operador>.Tipo
    if   (<Operador>.Tipo == 'F' && <Expresion2>.Tipo == 'I'){
        <Expresion2>.Tipo = 'F';
        <Expresion2>.Valor = Float(<Expresion2>.Valor);
     }
    if   (<Operador>.Tipo == 'F' && <Expresion3>.Tipo == 'I'){
        <Expresion3>.Tipo = 'F';
        <Expresion3>.Valor = Float(<Expresion3>.Valor);
    }
    switch (<Operador>.Tipo){
      'I':    <Expresion1>.Valor = Op_entera(<operador>.Clase,
                  <Expresion2>.Valor, <Expresion3>.Valor); break;
      'F':    <Expresion1>.Valor = Op_real(<operador>.Clase,
                  <Expresion2>.Valor, <Expresion3>.Valor); break;
    }
}
```


Gramáticas de atributos

- Las gramáticas de atributos se escriben en forma de tabla:
  - Las reglas gramaticales, a la izquierda
  - Las reglas semánticas asociadas, a la derecha

Regla gramatical	Regla semántica
Regla 1	Ecuaciones de atributo asociadas
…	…
Regla n	Ecuaciones de atributo asociadas

Gramáticas de atributos

- Ejemplo

Regla gramatical	Regla semántica
L → E n	`print(E.val)`
E → E + T	E_0`.val =` E_1`.val + T.val`
E → T	`E.val = T.val`
T → T * F	T_0`.val =` T_1`.val * F.val`
T → F	`T.val = F.val`
F → (E)	`F.val = E.val`
F → digito	`F.val = digito.valor_lexico`

Definiciones dirigidas por la sintaxis

- Cada símbolo gramatical tiene asociado un conjunto de atributos.
- El valor de un atributo en un árbol sintáctico se calcula mediante una regla semántica asociada a la producción utilizada en el nodo.
- Tipos de atributos:
  - **Sintetizados**: Su valor se calcula en función de atributos de nodos hijos en el árbol de análisis sintáctico.

```
A → aB  { A.atributo = a.atributo + B.atributo }
```

  - **Heredados**: Para un hijo se calculan a través de los atributos del padre y hermanos en el árbol de análisis sintáctico.

```
A → aB  { B.atributo = a.atributo – A.atributo }
```


Definiciones dirigidas por la sintaxis

- Las reglas semánticas establecen las dependencias entre los atributos
  - Estas dependencias se representan en un grafo.
- Del grafo de dependencias se obtiene el orden de evaluación de las reglas semánticas.
- Árbol sintáctico con anotaciones: árbol sintáctico que muestra información en cada nodo sobre los atributos.

Definiciones dirigidas por la sintaxis

- Forma de una *definición dirigida por la sintaxis*:
  - Cada producción $A \rightarrow \alpha$ tiene una o más reglas semánticas asociadas
  - Cada regla tiene la forma $b = f(c_1, c_2, .., c_n)$
  - b, que depende de $c_1, c_2, ..., c_n$, puede ser:
    - Un atributo sintetizado de A.
    - Un atributo heredado de uno de los símbolos del lado derecho de la producción.
  - Las funciones f de las reglas se escriben como expresiones.

Definiciones S-atribuidas

- Gramática S-atribuida: todos los atributos asociados con los símbolos gramaticales son sintetizados.

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

- Las reglas de evaluación de los atributos deben definirse en función de los atributos asociados con los símbolos gramaticales “hijos”.

Evaluación de atributos sintetizados

- Los valores de los atributos S se pueden calcular fácilmente mediante un recorrido ascendente (post-orden) del árbol sintáctico:

```
procedimiento EvaluarSintetizado(A:arbolSintactico){
  Para cada hijo H de A hacer
    EvaluarSintetizado(H);
  Calcular atributos sintetizados de A
}
```


Gramáticas S-atribuidas

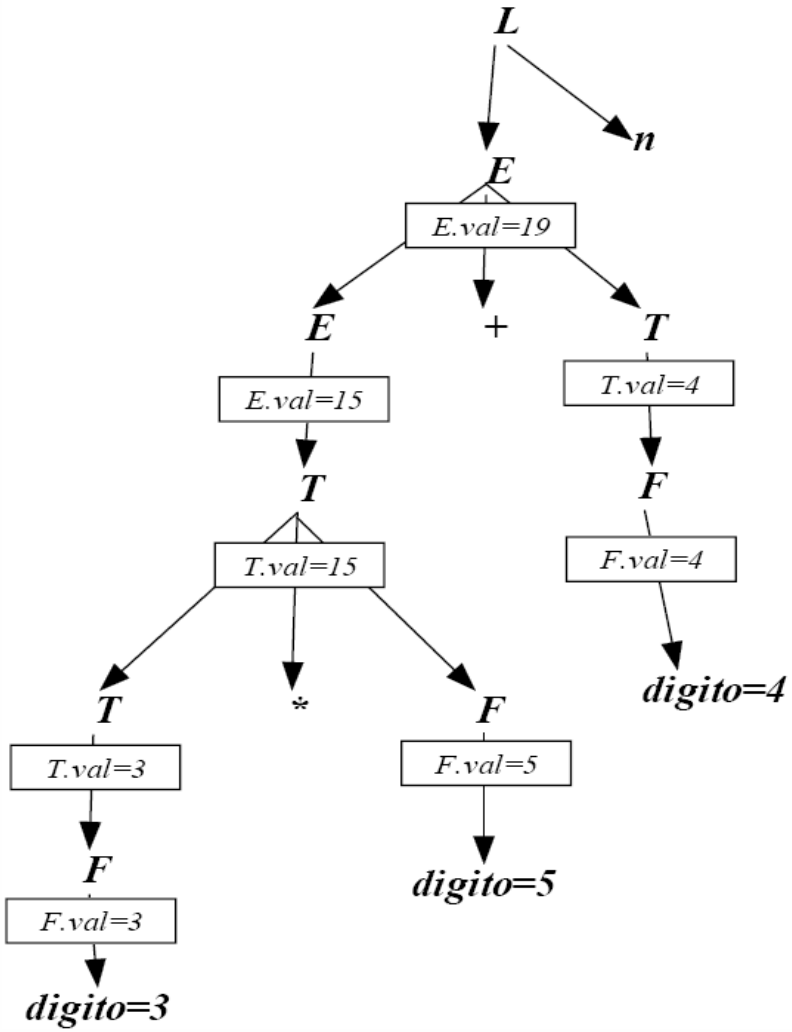
- Ejemplo de gramática S-Atribuida:
  - **Calculadora aritmética sencilla**. Se desea evaluar expresiones a la vez que las analizamos. Sea el conjunto de producciones y acciones siguientes:

L → E n	**{ print (E1.val) } (* n = salto línea *)**
E → E + T	**{ E0.val = E1.val + T3.val }**
E → T	**{ E0.val = T1.val }**
T → T * F	**{ T0.val = T1.val * F3.val }**
T → F	**{ T0.val = F1.val }**
F → (E)	**{ F0.val = E2.val }**
F → digito	**{ F0.val = digito }**

Gramáticas S-atribuidas

- Evaluación de la expresión “3 * 5 + 4”
- Resultado: se imprime el resultado de calcular 3 * 5 + 4.

L → E n
E → E + T | T
T → T * F | F
F → (E) | digito



Gramáticas L-atribuidas

- Gramática L-atribuida: todos los atributos asociados con los símbolos gramaticales son sintetizados o heredados pero cumpliendo que su evaluación dependa de los atributos asociados con los símbolos precedentes en la derivación:
  - Heredan del nodo "padre"
    - Ej: **`A→XYZ { Y.valor = A.valor }`**
  - Heredan de hermanos a su "izquierda"
    - Ej: **`A→XYZ { Y.valor = X.valor }`**
  - Heredan de otros atributos del mismo símbolo
    - Ej: **`A→XYZ { Y.valor = float(Y.int_value)*2 }`**

Gramáticas L-atribuidas

- Los atributos heredados permiten expresar la dependencia de una construcción del lenguaje con respecto al contexto en que aparece.

- Ejemplos:
  - Saber si un identificador está en la parte izquierda (dirección) o derecha (valor) de una expresión.
  - Conocer la posición de un argumento de función f(x,y,z) "¿Qué posición ocupa dentro de la lista de argumentos el argumento **y**?"

- Para su evaluación el análisis óptimo es el descendente.

Evaluación de atributos heredados

- Los valores de los atributos heredados se pueden calcular mediante un recorrido descendente (pre-orden) del árbol sintáctico:

```
procedimiento EvaluarHeredado(A:arbolSintactico){
  Para cada hijo H de A hacer{
    Calcular atributos heredados de H
    EvaluarHeredado(H);
  }
}
```


Evaluación de atributos heredados

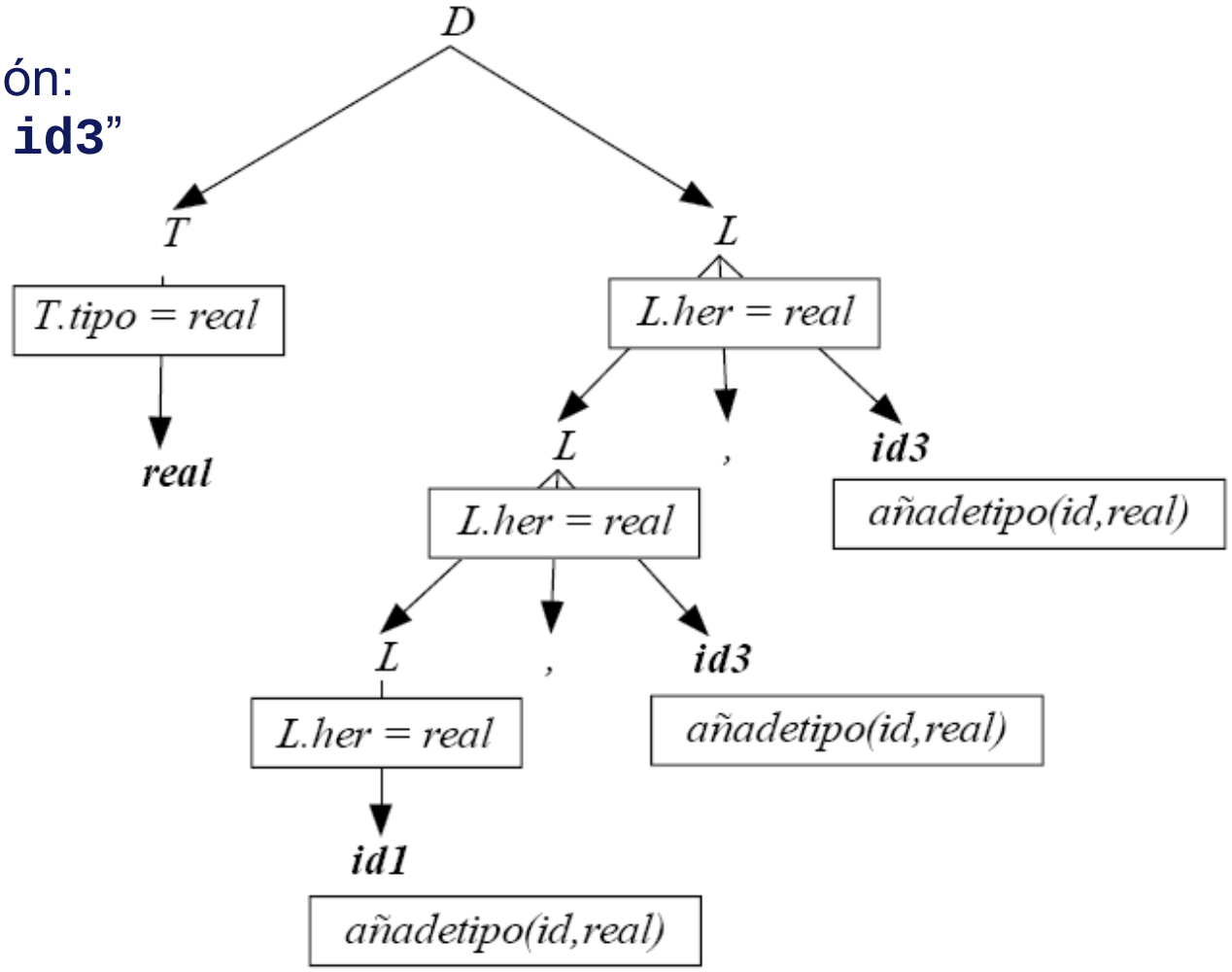
- Ejemplo:

D → T L	**{ L.her = T.tipo; }**
T → int	**{ T.tipo = entero; }**
T → real	**{ T.tipo = real; }**
L → L , id	**{ L_1.her = L_0.her; añadetipo(id,L_0.her); }**
L → id	**{ añadetipo(id, L.her); }**

Gramáticas L-atribuidas

- Evaluar la expresión: "**`real id1,id2,id3`**"

D → T L
T → int
T → real
L → L , id
L → id



Esquemas de traducción

- **Esquema de Traducción:** *gramática con atributos* cuyas acciones semánticas se expresan entre llaves.

- Notación complementaria donde las acciones se encuentran o bien intercaladas entre los símbolos de la parte derecha de las producciones, o bien al final de las mismas.

  – Inserción de acciones semánticas al final de cada producción.

```
S ::= B1 B2 { B1.atr = 1; B2.atr = 2; }
B ::= x { print(B.atr); }
```

  – Acciones semánticas intercaladas

```
Proced ::= procedure {CrearAmbito();} id Args Decl Sentencias;
```


Grafo de dependencias

- *Grafo de Dependencias*: para calcular el valor de un atributo es necesario calcular en primer lugar los valores de los atributos de los que depende, estableciendo una dependencia entre atributos.

- Cuando aparecen definidos atributos sintetizados y heredados, es necesario establecer un orden de evaluación.

- Orden de evaluación:
  - Para cualquier acción semántica de la forma:

```
X.atr = f(Y1.atr, ..., Yn.atr)
```

  - Los valores de los atributos **`Y1.atr, ..., Yn.atr`** deben estar disponibles antes de ejecutarla.

Grafo de dependencias

- El grafo de dependencias es un grafo dirigido acíclico:
  - *Un nodo para cada atributo*
  - *Un arco b → c si el atributo c depende del atributo b*

- Se construye con el siguiente algoritmo:

  - Nodos: Para cada nodo **n** del árbol sintáctico, hacer:
    - Para cada atributo **a** asociado al símbolo gramatical del nodo **n**, construir un nodo etiquetado con **a** en el grafo de dependencias.

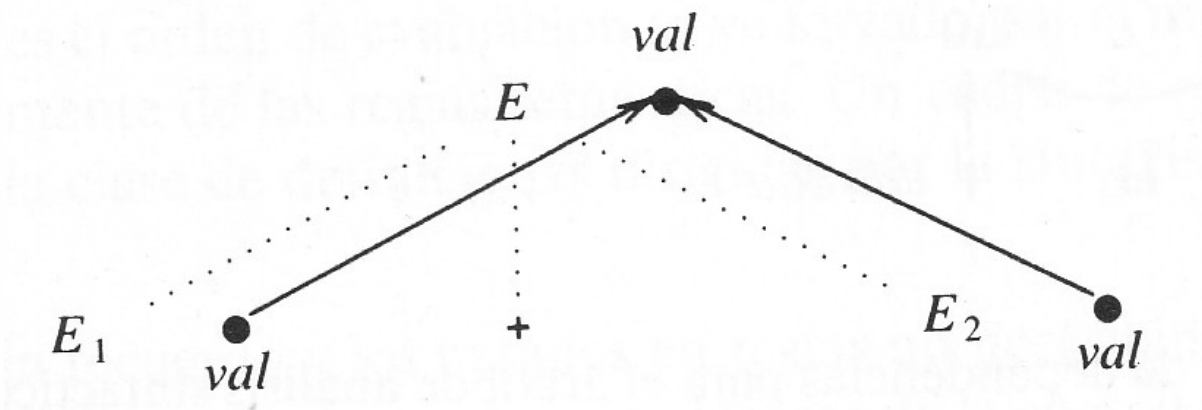
  - Arcos: Para cada nodo **n** del árbol sintáctico, hacer:
    - Para cada regla semántica **b = f (c1, c2, ..., cn)** asociada con la producción del nodo **n**, trazar arcos desde cada **ci** hasta **b**.

  - Los atributos sintetizados se representan marcando el nodo así: ●

  - El árbol sintáctico se representa en paralelo mediante líneas punteadas.

Grafo de dependencias

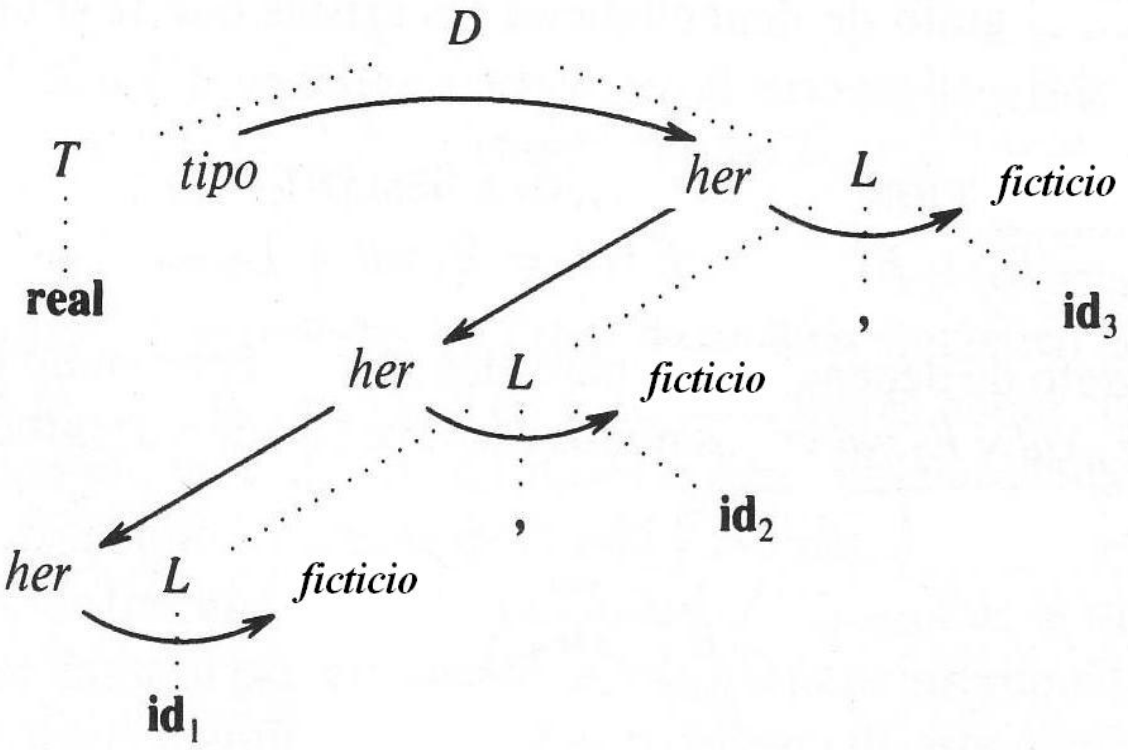
$E \rightarrow E + E$ $\quad$ $\{ E_0.val = E_1.val + E_2.val \}$



Grafo de dependencias

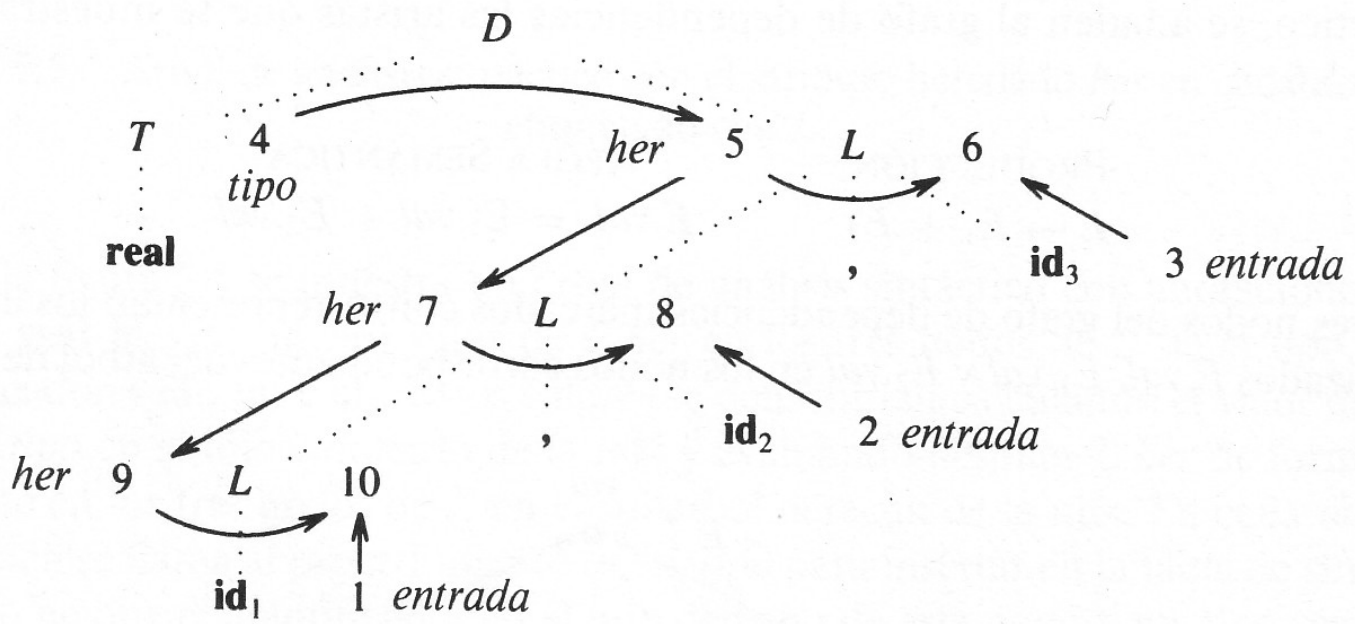
D → T L `{ L2.her = T1.tipo; }`
T → int `{ T0.tipo = entero; }`
T → real `{ T0.tipo = real; }`
L → L , id `{ L1.her = L0.her; añadetipo(id, L0.her);}`
L → id `{ añadetipo(id, L0.her); }`

Entrada:
`real id1,id2,id3;`



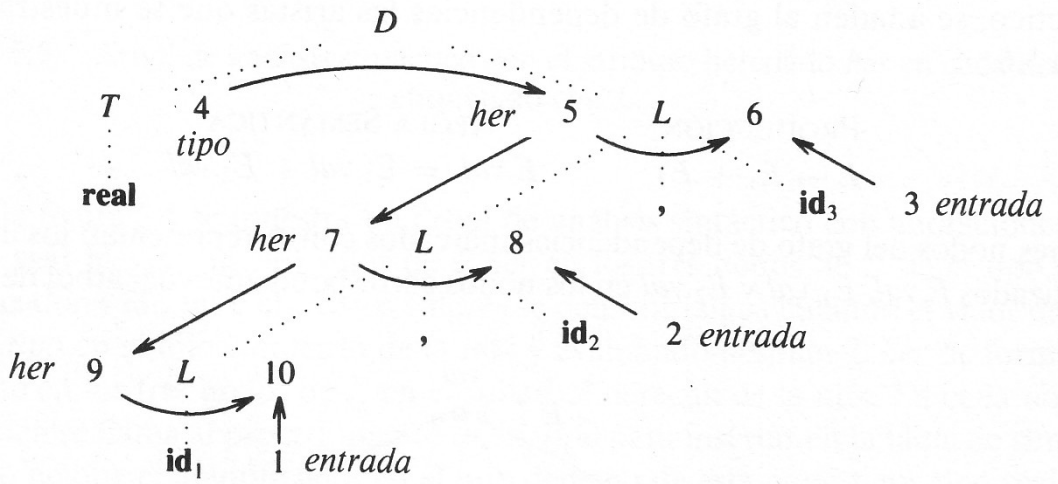
Orden de evaluación

- Clasificación topológica: orden de evaluación de las reglas semánticas asociadas a cada nodo del árbol de análisis sintáctico.
  - Se etiqueta cada nodo con un número
  - Los atributos independientes se evalúan antes que los dependientes
  - El grafo de dependencias debe ser acíclico



Orden de evaluación

- Una clasificación topológica se utiliza para generar un programa con el conjunto de reglas de evaluación ordenadas:




```
a4 = real;
a5 = a4;
añadetipo (id3.entrada,a5);
a7 = a5;
añadetipo (id2.entrada,a7);
a9 = a7;
añadetipo (id1.entrada,a9);
```


Orden de evaluación

- Al método anterior se le conoce como Método de árbol de análisis gramatical.

- Inconvenientes:
  - La complejidad añadida que la construcción del grafo de dependencias supone para la compilación (se realiza en tiempo de compilación).
  - El método debe determinar si el grafo es acíclico (en tiempo de construcción).

- Alternativa: Método basado en reglas.
  - El escritor del compilador analiza la gramática y fija un orden de evaluación de atributos (en tiempo de construcción del compilador).
  - *Basado en reglas*: analiza las reglas semánticas y depende de ellas.
  - Sólo puede hacerse para gramáticas “completamente no circulares”.
  - Prácticamente todos los compiladores lo utilizan.

Evaluación ascendente

- Los principales métodos de análisis sintáctico procesan la entrada de izquierda a derecha, lo que implica que los atributos no pueden tener dependencias “hacia atrás”.
  - Este problema se plantea sólo para atributos heredados.
- Los analizadores ascendentes (LR) son más adecuados para manejar atributos sintetizados.
  - Reducen cuando se conoce toda la parte derecha de una producción.
- Es posible implementar traductores ascendentes para atributos heredados utilizando técnicas avanzadas.

Evaluación ascendente

- 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 justo antes de cada reducción.
- El analizador LR contiene una pila de valores adicional para almacenar los valores de los atributos sintetizados.
  - Si hay más de un atributo para un símbolo, se almacenan como estructuras.
- El *Analizador* es similar, pero ahora utiliza producciones compuestas por símbolos más acciones semánticas:
  - Al aplicar una reducción se realizan los cálculos indicados en las acciones semánticas, utilizando generalmente los elementos de la pila de valores.
  - Un desplazamiento consiste en la inserción de valores de *token* tanto en la pila de valores como en la pila de análisis sintáctico.

Evaluación de atributos

- Si una producción maneja un único atributo “val”, la pila contiene tanto los estados como los valores del atributo para los símbolos que ya han sido procesados (desplazados o reducidos anteriormente).

- Si un símbolo no tiene atributo “val” la entrada correspondiente en la tabla está sin definir.

- Acciones sobre la pila de valores:

  - Obtener un valor: **`Pila.pop(valor)`**
  - Descartar un valor: **`Pila.pop()`**
  - Insertar un valor: **`Pila.push(valor)`**

Evaluación de atributos

- Ejemplo:

	Pila de análisis sintáctico	Entrada	Acción de análisis sintáctico	Pila de valores	Acción semántica
1	\$	3*4+5 \$	desplazamiento	\$	
2	\$ **n**	*4+5 \$	reducción de $E \rightarrow$ **n**	\$ **n**	$E.val =$ **n** $.val$
3	\$ E	*4+5 \$	desplazamiento	\$ 3	
4	\$ E *	4+5 \$	desplazamiento	\$ 3 *	
5	\$ E * **n**	+5 \$	reducción de $E \rightarrow$ **n**	\$ 3 * **n**	$E.val =$ **n** $.val$
6	\$ E * E	+5 \$	reducción de $E \rightarrow E * E$	\$ 3 * 4	$E_1.val = E_2.val * E_3.val$
7	\$ E	+5 \$	desplazamiento	\$ 12	
8	\$ E +	5 \$	desplazamiento	\$ 12 +	
9	\$ E + **n**	\$	reducción de $E \rightarrow$ **n**	\$ 12 + **n**	$E.val =$ **n** $.val$
10	\$ E + E	\$	reducción de $E \rightarrow E + E$	\$ 12 + 5	$E_1.val = E_2.val + E_3.val$
11	\$ E	\$		\$ 17	

Evaluación de atributos

- Es posible calcular con este método (LR) atributos heredados de hermanos previamente calculados.
- Es necesario introducir una producción ϵ adicional.
- El valor del atributo se almacena en una variable auxiliar y se calcula en función del valor en la cima de la pila **antes** del reconocimiento de C.
- En Yacc/CUP:

```
A : B { aux = 2 * B.atr } C
```

Regla gramatical	Regla semántica
A → BC	`{ C.her = 2 * B.atr }`
B → …	`{ Calcular B.atr; }`
C → …	`{/* utilizar C.her */}`

Regla gramatical	Regla semántica
A → BXC	`{ C.her = 2 * aux }`
B → …	`{ Calcular B.atr; }`
X → Є	`{ aux = Pila.Cima() }`
C → …	`{/* C.her disponible */}`

Evaluación de atributos

- El cálculo de los atributos depende de la estructura de la gramática.
- Es posible simplificar el cálculo mediante una modificación de las reglas gramaticales.
- Teorema de Knuth: *Dada una gramática con atributos, todos los atributos heredados se pueden convertir en sintetizados modificando adecuadamente la gramática, sin cambiar el lenguaje.*
  - En la práctica no se utiliza demasiado, pues puede generar gramáticas y reglas semánticas más complejas que las originales.

Bibliografía

- *Básica:*

  - *Compiladores: principios, técnicas y herramientas*. A.V. Aho, R. Sethi, J.D. Ullman. Addison-Wesley Iberoamerica. 1990.

  - *Construcción de compiladores. Principios y práctica.* Kenneth C. Louden. Thomson-Paraninfo. 2004.