Universidad de Huelva

Tema 5

Análisis semántico

Índice

Índice

Objetivos:

- Verificar la semántica de la cadena de entrada:
  - Verificar que las variables han sido declaradas previamente
  - Comprobación de tipos en las expresiones
  - Comprobación de los parámetros de una función
  - Elección de la función u operador en caso de sobrecarga o polimorfismo
- Emitir informes de errores semánticos
- Construir el Árbol de Sintaxis Abstracta

Principio de la traducción dirigida por la sintaxis:

- El significado (semántica) de una frase está directamente relacionado con su estructura sintáctica

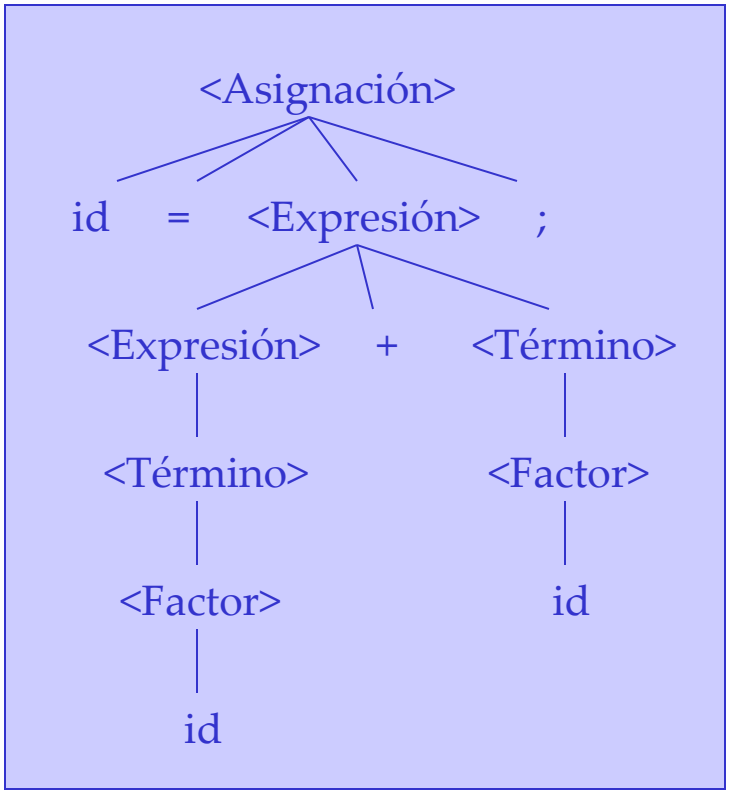
Especificación semántica

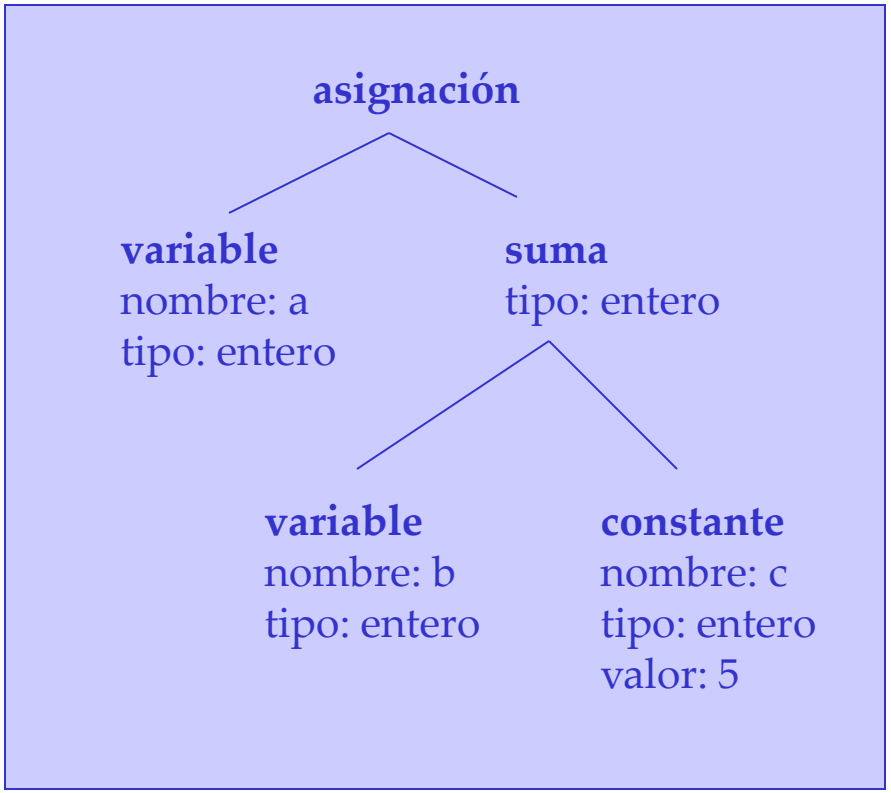
- Se inserta en la especificación sintáctica
- Se amplían las gramáticas libres de contexto:
  - Se asocia información a los símbolos (atributos)
  - Se asocian acciones semánticas (código) a las producciones

Árbol de Sintaxis Abstracta

- Recoge toda la información sintáctica y semántica
- Elimina los símbolos que no aportan significado
- Los nodos reflejan los datos asociados a cada símbolo
- El objetivo de las acciones semánticas es construir este árbol

Ejemplo: a = b + c ;


<Asignación>
id = <Expresión> ;
<Expresión> + <Término>
<Término> <Factor>
<Factor> id
id


asignación
variable
nombre: a
tipo: entero
suma
tipo: entero
variable
nombre: b
tipo: entero
constante
nombre: c
tipo: entero
valor: 5

Índice

Atributos:

- Elementos de los símbolos del lenguaje que almacenan información semántica:
  - Tipo de datos
  - Referencia a las variables
  - Etc.

Acciones semánticas

- Expresiones que permiten calcular el valor de los atributos

Tipos de atributos:

- **Atributos sintetizados:**
  - Aquellos cuyo valor se calcula a partir de la información de los nodos hijos.
  - Representan un flujo de información ascendente en el árbol.
  - Ej.: <A> → <B> <C> <D> { A.a = B.b + C.c + D.d }
- **Atributos heredados:**
  - Aquellos cuyo valor se calcula a partir de la información del nodo padre o de los hermanos.
  - Representan un flujo de información descendente.
  - Ej.: <A> → <B> <C> <D> { C.c = A.a + B.b }

- Los atributos sintetizados de un símbolo representan una información que se calcula después de reconocer sintácticamente ese símbolo.
- Los atributos heredados de un símbolo representan una información que es necesario conocer antes de reconocer sintácticamente el símbolo.
- Generalmente se considera que los atributos de los tokens (símbolos terminales) son siempre sintetizados. Estos atributos son los generados por el analizador léxico.

Ejemplo (declaración de variables en C):

- S → sintetizado
- h → heredado.

Atributos	Estructura
D.s, L.s, Lp.s	class L ista{ int tipo; String nombre; Lista sig; }
T.s, L.h, Lp.h	int

Producción	**Regla semántica asociada**
<D> → <T> <L> **pyc**	{ L.h = T.s; D.s = L.s }
<T> → **float**	{ T.s = REAL }
<T> → **int**	{ T.s = ENTERO }
<L> → **id** <Lp>	{Lp.h = L.h; L.s.tipo = L.h; L.s.nombre = id.lexema; L.s.sig = Lp.s}
<Lp> → **coma id** <Lp_1>	{ Lp_1.h = Lp.h; Lp.s.tipo = Lp.h; Lp.s.nombre = id.lexema; Lp.s.sig = Lp_1.s }
<Lp> → λ	{ Lp.s = null }

Índice

Gramáticas con atributos por la izquierda

- Son un subconjunto de las gramáticas con atributos.
- Los atributos heredados se calculan exclusivamente a partir de los símbolos a la izquierda del atributo.
- Cualquier información heredada está disponible en el momento de ejecutar la reglas de producción correspondiente.
- Los flujos de información en el árbol de sintaxis abstracta se producen de izquierda a derecha, de arriba hacia abajo o de abajo hacia arriba. Nunca de derecha a izquierda.
- Permiten definir traductores de una sola pasada.

Esquemas de traducción dirigidos por la sintaxis (ETDS)

- Son un formalismo para construir traductores de una sola pasada
- Emplean gramáticas con atributos por la izquierda.
- Incluyen acciones semánticas encerradas entre llaves ({ ... }) en cualquier punto de las reglas de producción.
- Las acciones semánticas incluyen fragmentos de código en un lenguaje de programación.
- Las acciones semánticas pueden incluir instrucciones que no se refieran al cálculo de atributos.

Índice

Desarrollo de un ETDS basado en análisis descendente

- Los atributos sintetizados de cada símbolo S se almacenan en la estructura de datos (objeto) que devuelve la función asociada al símbolo S.
- Los atributos heredados de cada símbolo S se describen como parámetros de la función asociada al símbolo S.
- Las acciones semánticas incluidas en las reglas de producción de un símbolo S se añaden directamente al código de la función asociada al símbolo S.

Ejemplo de un ETDS basado en análisis descendente

- Reglas de producción
  - <D> → <T> <L> **pyc**
  - <T> → **float**
  - <T> → **int**
  - <L> → **id** <Lp>
  - <Lp> → **coma id** <Lp_1>
  - <Lp> → λ
- Atributos
  - clase L { int tipo; String id; L siguiente; }
  - T.s (int) (sintetizado), L.h (int) (heredado), Lp.h (int heredado)
  - D.s (L) (sintetizado), L.s (L) (sintetizado), Lp.s (L) (sintetizado)

Ejemplo de un ETDS basado en análisis descendente

- Gramática atribuida
  - <D> → <T> { *L.h = T.s* } <L> **pyc** { *D.s = L.s* }
  - <T> → **float** { *T.s = REAL* }
  - <T> → **int** { *T.s = ENTERO* }
  - <L> → { *L.s = new L();* } **id** { *L..s.id = id; L.s.tipo = L.h; Lp.h = L.h* }
    <Lp> { *L..s.siguiente = Lp.s* }
  - <Lp> → { *Lp.s = new L();* } **coma id** { *Lp.s.id = id; Lp.s.tipo = Lp.h;* Lp_1*.h = Lp.h* }
    <Lp_1> { *Lp.s.siguiente =* Lp_1*.s* }
  - <Lp> → λ { *Lp.s = null* }

Ejemplo de un ETDS basado en análisis descendente

- Función asociada a <D>

```
L parseD() {
  L D_s;
  switch(nextToken.getKind()){
   case FLOAT:
   case INT:
     int T_s = parseT();     // las funciones devuelven objetos
     L L_s = parseL( T_s ); // los atributos heredados
     match(PYC);            // se pasan como argumentos
     D_s = L_s;
     break;
   default: error();
  }
  return D_s;
}
```

- Función asociada a <T>

```
int parseT() {
  int T_s;
  switch(nextToken.getKind()) {
    case FLOAT:
      match(FLOAT);
      T_s = REAL;
      break;
    case INT:
      match(INT);
      T_s = ENTERO;
      break;
    default: error();
  }
  return T_s;
}
```

- Función asociada a <L>

```
L parseL(int L_h) {
  L L_s;
  switch(nextToken.getKind()) {
    case ID:
       L_s = new L();
       Token tk = match(ID); // El analizador léxico devuelve objetos
       L_s.tipo = L_h; L_s.id = tk.lexema;
       L Lp_s = parseLp(L_h);
       L_s.siguiente = Lp_s;
       break;
    default: error();
  }
  return L_s;
}
```

- Función asociada a <Lp>

```
L parseLp(int Lp_h) {
   L Lp_s;
   switch(nextToken.getKind()) {
    case COMA: Lp_s = new L();
               match(COMA);
               Token tk = match(ID);
               Lp_s.tipo = Lp_h;  Lp_s.id = tk.lexema;
               L Lp1_s = parseLp(Lp_h);
               Lp_s.siguiente = Lp1_s;
               break;
    case PYC:  Lp_s = null; break;
    default: error();
   }
   return Lp_s;
 }
```

Índice

Desarrollo de un ETDS basado en análisis ascendente

- A cada estado de la tabla se le puede asociar un símbolo (terminal o no terminal) que corresponde al símbolo que provoca la transición a ese estado.
- Los atributos sintetizados de cada símbolo S se almacenan en una estructura de datos (objeto).
- Se utiliza una pila de atributos, paralela a la pila de estados, para almacenar estas estructuras.
- Las acciones semánticas situadas al final de una regla se ejecutan al reducir la regla.

Desarrollo de un ETDS basado en análisis ascendente

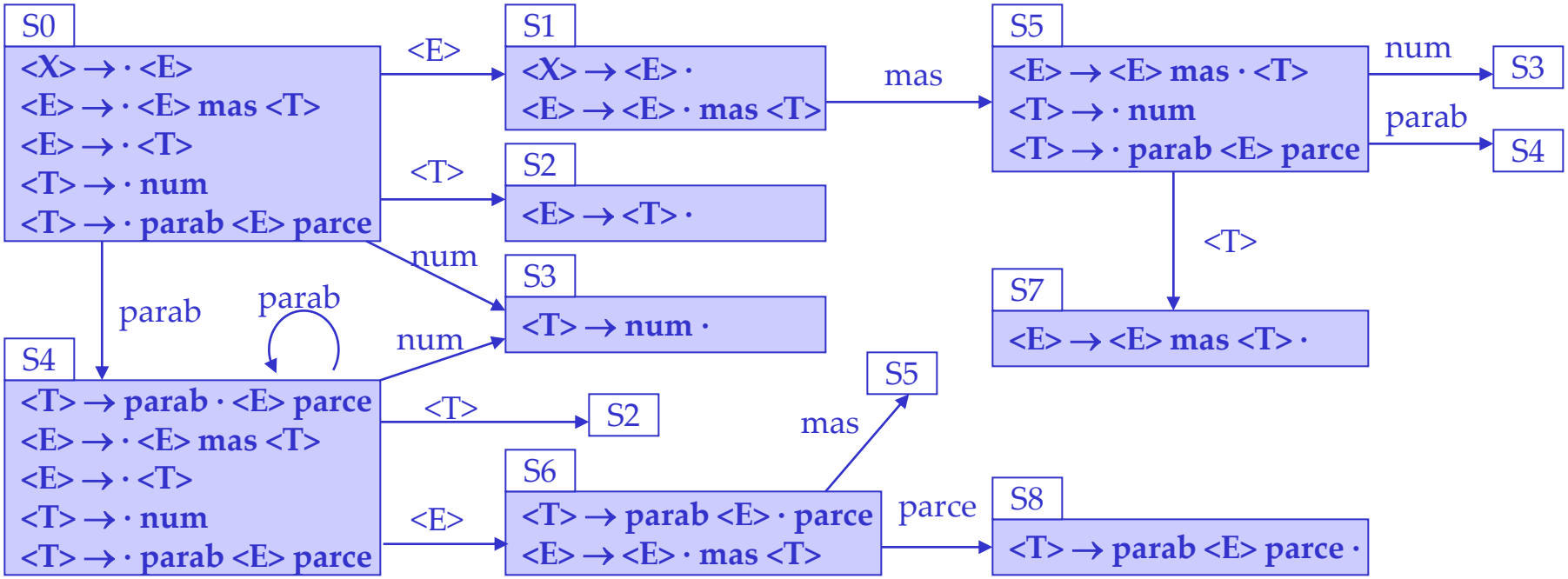
- Las acciones semánticas situadas en puntos intermedios se sustituyen por símbolos ficticios (marcadores) a los que se asocia una regla de producción λ seguida de la acción semántica.
- Conocida la longitud de una regla, se puede saber la posición de cada estructura de datos tomando como base la cima de la pila.
- Los atributos heredados de un símbolo no se pueden tratar directamente, sino que se introducen como atributos sintetizados del símbolo anterior.

Ejemplo de un ETDS basado en análisis ascendente

- Reglas de producción
  - <E> → <E> **mas** <T>
  - <E> → <T>
  - <T> → **num**
  - <T> → **parab** <E> **parce**
- **Ejemplo**
  - **7 + (8 + 2 + (3 + 1))**

Ejemplo de un ETDS basado en análisis ascendente

- Autómata Reconocedor de Prefijos Viables



Ejemplo de un ETDS basado en análisis ascendente

- Tabla de análisis SLR

Símbolo	Estado	mas	num	parab	parce	$	<E>	<T>
-	0		d3	d4			1	2
<E>	1	d5				aceptar		
<T>	2	r2			r2	r2		
num	3	r3			r3	r3		
parab	4		d3	d4			6	2
mas	5		d3	d4				7
<E>	6	d5			d8			
<T>	7	r1			r1	r1		
parce	8	r4			r4	r4		

Ejemplo de un ETDS basado en análisis ascendente

- Traza de la cadena 7+5

Pila de atributos	Pila de estados	Entrada	Acción
-	0	num mas num $	d3
- num	0 3	mas num $	r3
- <T>	0 2	mas num $	r2
- <E>	0 1	mas num $	d5
- <E> mas	0 1 5	num $	d3
- <E> mas num	0 1 5 3	$	r3
- <E> mas <T>	0 1 5 7	$	r1
- <E>	0 1	$	aceptar

Ejemplo de un ETDS basado en análisis ascendente

- Gramática atribuida
  - <E> → <E> **mas** <T> { *$$.op = SUMA; $$.izquierda = $1; SS.derecha = $3;* }
  - <E> → <T> { *$$.op = TERMINO; $$.izquierda = $1; $$.derecha = NULL;* }
- Notación (utilizada en YACC)
  - $$ es la estructura de datos de la parte izquierda de la regla.
  - $1 .. $N es la estructura de datos del símbolo 1 .. N de la parte derecha.
  - Para una regla de N símbolos, la estructura $M se encuentra en la posición N-M desde la cima de la pila.
  - No admite atributos heredados.

Ejemplo de un ETDS basado en análisis ascendente

- Notación
  - Las acciones semánticas situadas al final de una regla se ejecutan al reducir esa regla
  - Las acciones semánticas intermedias se sustituyen por nuevas reglas (marcadores). Por ejemplo:
    - <E> → <E> { *$$.izq = $1;* } **mas** { *$$.op = SUMA;* } <T> { *$$.dcha = $3;* }
  - se transforma en:
    - <E> → <E> <M1> **mas** <M2> <T> { *$$.dcha = $3;* }
    - <M1> → λ { *$$.izq = $1;* }
    - <M1> → λ { *$$.op = SUMA;* }

Índice

Generación de una estructura de lista

- Gramática LL(1) BNF
  - <Lista> → **id** <SigueLista>
  - <SigueLista> → **coma id** <SigueLista>
  - <SigueLista> → λ
- Gramática LL(1) EBNF
  - <Lista> → **id** (**coma id**)*
- Gramática LR(1)
  - <Lista> → **id**
  - <Lista> → <Lista> **coma id**

Generación de una estructura de lista

- Gramática LL(1) BNF
  - <Lista> → **id** <SigueLista>

    { e = new Element(), e.id = id, e.next = SigueLista.s, Lista.s = e }
  - <SigueLista> → **coma** **id** <SigueLista$_1$>

    { e = new Element(), e.id = id, e.next = SigueLista$_1$.s, SigueLista.s = e }
  - <SigueLista> → λ { SigueLista.s = null }
- Gramática LL(1) EBNF
  - <Lista> → **id** { e = new Element(), e.id = id, e.next = null, Lista.s = e, last = e }

    (**coma** **id** { e = new Element(), e.id = id, e.next = null ,

    last.next = e, last = last.next })*

Generación de una estructura de lista

- Gramática LR(1)
  - <Lista> → **id**

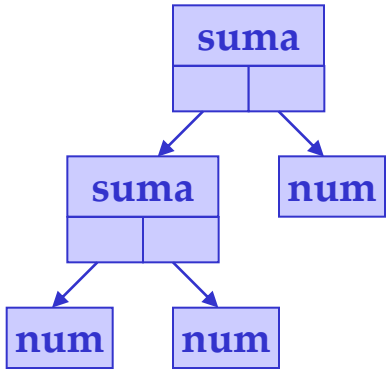
    { e = new Element(), e.id = id, e.next = null, Lista.s = e, Lista.last = e }
  - <Lista> → <Lista$_1$> **coma id**

    { e = new Element(), e.id = id, e.next = null, Lista.s = Lista$_1$.s,

    Lista$_1$.last.next = e, Lista.last = e }
- Gramática LR(1)
  - <Lista> → **id** { e = new Element(), e.id = id, e.next = null, Lista.s = e }
  - <Lista> → **id coma** <Lista$_1$>

    { e = new Element(), e.id = id, e.next = Lista$_1$.s, Lista.s = e }

Generación de una estructura de árbol binario



- Gramática LL(1) BNF
  - <Árbol> → **num** <SigueÁrbol>
  - <SigueÁrbol> → **más num** < SigueÁrbol >
  - < SigueÁrbol > → λ
- Gramática LL(1) EBNF
  - <Árbol> → **num** (**más num**)*
- Gramática LR(1)
  - <Árbol> → **num**
  - < Árbol > → < Árbol $_1$> **más num**

Generación de una estructura de árbol binario

- Gramática LL(1) BNF
  - <Árbol> → **num** { SigueÁrbol.h = num }
    <SigueÁrbol> {Árbol.s = SigueÁrbol.s }
  - <SigueÁrbol> → **más num** { $SigueÁrbol_1$.h = new Suma(SigueÁrbol.h,num) }
    <$SigueÁrbol_1$> { SigueÁrbol.s = $SigueÁrbol_1$.s }
  - <SigueÁrbol> → λ { SigueÁrbol.s = SigueÁrbol.h }

- Gramática LL(1) EBNF
  - <Árbol> → **num** { Árbol.s = num }
    (**más num** { Árbol.s = new Suma(Árbol.s,num) })*

Generación de una estructura de árbol binario

- Gramática LR(1)
  - <Árbol> → **num** { Árbol.s = num }
  - <Árbol> → <Árbol$_1$> **más** **num** { Árbol.s = new Suma(Árbol$_1$.s, num) }

Generación de una estructura de vector

class Vector { public Vector(); public add(Element e); }

- Gramática LL(1) BNF
  - <Lista> → **num** <SigueLista>
    { v = new Vector(), v.add(num), SigueLista.h = v, Lista.s = v }
  - <SigueLista> → **más num**
    { SigueLista.h.add(num), $SigueLista_1$.h = SigueLista.h }
    <$SigueLista_1$>
  - <SigueLista> → λ { }
- Gramática LL(1) EBNF
  - <Lista> → { v = new Vector() } **num** { v.add(num) }
    (**más num** { v.add(num) })* { Lista.s = v }

Generación de una estructura de vector

class Vector { public Vector(); public add(Element e); }

- Gramática LR(1)
  - <Lista> → **num** { v = new Vector(), v.add(num), Lista.s = v }
  - <Lista> → <Lista$_1$> **más num** { Lista.s = Lista$_1$.s, Lista.s.add(num) }