ANÁLISIS SEMÁNTICO

El análisis semántico dota de un significado coherente a lo que hemos hecho en el análisis sintáctico. El chequeo semántico se encarga de que los tipos que intervienen en las expresiones sean compatibles o que los parámetros reales de una función sean coherentes con los parámetros formales

FUNCIONES PRINCIPALES

- ü Identificar cada tipo de instrucción y sus componentes
- ü Completar la Tabla de Símbolos
- ü Realizar distintas comprobaciones y validaciones:
  - Comprobaciones de tipos.
  - Comprobaciones del flujo de control.
  - Comprobaciones de unicidad.
  - Comprobaciones de emparejamiento.

El Analizador Semántico finaliza la fase de Análisis del compilador y comienza la fase de Síntesis, en la cual se comienza a generar el código objeto.

La especificación de la semántica puede realizarse de dos formas:

- ü Lenguaje natural
- ü Especificación formal: Semántica Operacional, semántica denotacional, semántica Axiomática, Gramáticas con Atributos.

ACCIONES SEMÁNTICAS

Dependiendo del tipo de sentencias, las acciones semánticas pueden agruparse en:

- ü Sentencias de Declaración: Completar la sección de tipos de la Tabla de Símbolos.
- ü Sentencias "ejecutables": Realizar comprobaciones de tipos entre los operandos implicados.
- ü Funciones y procedimientos: Comprobar el número, orden y tipo de los parámetros actuales en cada llamada a una función o procedimiento.

- Identificación de variables: Comprobar si un identificador ha sido declarado antes de utilizarlo.
- Etiquetas: Comprobar si hay etiquetas repetidas y validación.
- Constantes: Comprobar que no se utilicen en la parte izquierda de una asignación.
- Conversiones y equivalencias de tipo: Verificación.
- Sobrecarga de operadores y funciones: Detectar y solventar.

GRAMÁTICAS CON ATRIBUTOS

Una Gramática con Atributos es una generalización de las Gramáticas Libres de Contexto, denominada Definición Dirigida por la Sintaxis:

- Cada símbolo gramatical puede tener asociado un conjunto finito de atributos, que pueden ser de los siguientes tipos:
  - Sintetizados: su valor se calcula en función de los atributos de los nodos hijos.
  - Heredados: su valor se calcula en función de los atributos de los nodos hermanos y/o del nodo padre.
  - Cada atributo tomará valores en un dominio.
  - Cada producción llevará asociadas un conjunto de reglas semánticas.
  - Las relaciones de dependencia entre atributos, establecidas por las reglas semánticas, se representarán mediante el Grafo de Dependencias.

A partir de estas gramáticas se llevan a cabo las denominadas “Traducciones dirigidas por sintaxis”.

Atributos Sintetizados

En el caso de los símbolos terminales de la gramática, su atributo no es más que el lexema asociado al token reconocido por el analizador léxico.

Una gramática con atributos se denomina Gramática S-Atribuida si todos los atributos son sintetizados. Siempre es posible transformar una Gramática con Atributos en Gramática S-Atribuida.

Ejemplo: Analizar la forma sentencial 12+3*6, a partir de la siguiente definición dirigida por la sintaxis:

Producción	*Regla semántica*
$E_1 \rightarrow E_2 + T$	$E_1.val = E_2.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T_1 \rightarrow T_2 * F$	$T_1.val = T_2.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow num$	$F.val = valor(num)$

Atributos Heredados

Una gramática con atributos se denomina Gramática L-Atribuida si cada atributo que se evalúa cumple una de las siguientes condiciones:

- ü Es un atributo sintetizado
- ü Dada una producción A à $X_1X_2..X_j..X_n$, el atributo heredado asociado a X_j depende únicamente de los atributos de $X_1,...,X_{j-1}$ y/o de atributos heredados asociados al símbolo A.

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, para lo cual se deberá establecer una dependencia entre atributos mediante un Grafo de Dependencias.

Ejemplo: Analizar la forma sentencial real id1, id2, id3 a partir de la siguiente definición dirigida por la sintaxis:

Producción	*Reglas semánticas*
$D \rightarrow T\ L$	L.her = T.tipo
$T \rightarrow$ int	T.tipo = integer
$T \rightarrow$ real	T.tipo = real
$L \rightarrow L_1$, id	L_1.her = L.her AñadeTipoTS(id.entrada, L.her)
$L \rightarrow$ id	AñadeTipoTS(id.entrada, L.her)

Ejemplo: Dada la gramática G = ({L, A}, {a}, {L à AL | A, A à a}, L), obtener una gramática con atributos que al analizar una cadena calcule el número de a's que la componen.

Ejemplo: Dada la gramática G = ({L, E, R}, {“,”, id, [,]}, {L à id | id[E], E à R| E, R , R à id}, L), definir un atributo denominado num_dimensiones y las reglas semánticas asociadas a cada producción, de forma que al analizar una sentencia obtengamos el número de dimensiones que tiene el array referenciado.

Ejemplo: Dada la gramática G = ({S, L, B}, {0, 1, .}, {S à L.L | L, L à LB | B, B à 0 | 1}, S), obtener una gramática con atributos que al analizar un número en binario calcule su equivalente en decimal.

Evaluación de Atributos con Analizadores Sintácticos Descendentes LL(1)

Las Gramáticas L-Atribuidas engloban a la mayoría de las gramáticas con atributos basadas en gramáticas LL(1).

Se define Esquema de Traducción como una gramática con atributos cuyas acciones semánticas se expresan entre llaves, y se encuentran intercaladas entre los símbolos de la parte derecha de las producciones asociadas a la gramática, o bien al final de las mismas.

Para realizar el Análisis Sintáctico Descendente de atributos con una gramática LL(1) será necesario transformar dicha gramática a un Esquema de Traducción en el que se insertarán las acciones semánticas necesarias, teniendo en cuenta que un valor de un atributo debe estar calculado antes de poder ser utilizado.

El Analizador es similar, pero ahora trabajará con producciones compuestas de los símbolos más las acciones semánticas:

a) Al aplicar una producción introduciremos en la pila los símbolos y las acciones semánticas.

b) Cuando en el tope de la pila se encuentre una acción semántica, pasaremos a ejecutarla, eliminándola a continuación del tope.

Ejemplo: Dada la gramática G = {{E, T}, {+, -, (,), num}, {E à E+T | E-T | T, T à (E) | num}, E}:

a) Obtener un Esquema de Traducción que permita conocer el resultado de una operación válida para dicha gramática.

b) Aplicar un Análisis Sintáctico Descendente de atributos sobre el Esquema de Traducción obtenido para analizar la evaluación de la expresión 6-3+4.

Ejemplo: Dada la gramática G = {{E, T, R}, {+, -, (,), num}, {E à TR, R à +TR | -TR | ξ, T à (E) | num}, E}:

a) Obtener un Esquema de Traducción que permita conocer el resultado de una operación válida para dicha gramática.

b) Aplicar un Análisis Sintáctico Descendente de atributos sobre el Esquema de Traducción obtenido para analizar la evaluación de la expresión 6-3+4.

Evaluación de Atributos con Analizadores Sintácticos Ascendentes LR(1)

En los analizadores LR no se conoce la producción aplicada hasta que no se ha analizado la parte derecha, por lo que las acciones semánticas deberán aplicarse al final de la producción.

En las Gramáticas S-Atribuidas las producciones con sus correspondientes acciones semánticas presentan la siguiente forma:

A à A1A2...An {acciones semánticas}

En las Gramáticas L-Atribuidas pueden presentarse de esta otra forma:

A à {0} A1 {1} A2 {2} ... An {n}

Donde {0}, {1}, ... , {n} son acciones semánticas.

Para poder trabajar con ellas se han de transformar de la siguiente manera:

A à S0 A1 S1 A2 S2 … An {n} S0 à ξ {0}
S1 à ξ {1}
….
Sn-1 à ξ {n-1}

Donde S0, S1, …, Sn son nuevos símbolos no terminales.

Para la realización del análisis, los atributos pueden almacenarse en una pila adicional, o bien en la misma pila.

Para la gramática G = ({E, T}, {+, -, (,), num}, {E à E+T | E-T | T, T à (E) | num}, E), podríamos realizar una implementación de las acciones semánticas asociadas de la siguiente manera, para un analizador LR:

Producción	*Fragmento de código*
L → E (amp. LR)	Print(val[tope])
E_1 → E_2 + T	val[nuevoTope] = val[tope-2]+val[tope]
E → T	
T_1 → T_2 * F	val[nuevoTope] = val[tope-2]*val[tope]
T → F	
F → (E)	val[nuevoTope] = val[tope-1]
F → num	

Los fragmentos de código se ejecutarán justo antes de que tenga lugar una reducción por la regla asociada.

COMPROBACIÓN DE TIPOS

Aspectos Generales

Un lenguaje con comprobación fuerte de tipos es capaz de garantizar que los programas se pueden ejecutar sin errores de tipo, por lo que los errores de tipo se detectarán siempre en tiempo de compilación.

Como mínimo, ante un error, un comprobador de tipos debe informar de la naturaleza y posición del error y recuperarse para continuar con la comprobación del resto del programa a analizar.

Veamos algunas de las operaciones a tener en cuenta en una comprobación de tipos:

- Conversión de Tipos: A veces es necesario transformar el tipo de una expresión para utilizar correctamente un operador o para pasar de forma adecuada un parámetro a una función.

- Coerción: Es una conversión de tipos que realiza de forma implícita el propio compilador. Si es el programador el que realiza la conversión se tratará entonces de una conversión explícita.

- Sobrecarga de operadores: La sobrecarga se resuelve determinando el tipo de cada una de las expresiones intervinientes en la sobrecarga.

- Funciones polimórficas: Son aquellas que trabajan con argumentos cuyo tipo puede cambiar en distintas llamadas a la función.

Especificación de un Comprobador de Tipos Básico

Básicamente se deberán realizar dos tareas:

1. Asignación de tipos: En las declaraciones.

2. Evaluación y comprobación de tipos: En las expresiones y en las funciones, así como en las sentencias.

Sea la gramática:

P à D ; S
D à D ; D | id: T
T à char | entero | real | booleano | array[num] of T | ^T | T à T
S à id := E | if E then S | while E do S | S;S
E à literal | num | id | id[E] | id^ | E op_lógico E | E op_arit E | E mod E | id(E)

Símbolo	Atributo	Dominio
id	$id_{.tipo}$	{entero, real, char, booleano, puntero(), array(,), funcion, error_tipo}
	$id_{.entrada}$	Números enteros (posición en TS)
num	$num_{.val}$	Números enteros
E	$E_{.tipo}$	{entero, real, char, booleano, puntero(), array(,), funcion, error_tipo}
T	$T_{.tipo}$	{entero, real, char, booleano, puntero(), array(,), funcion, error_tipo}
S	$S_{.tipo}$	{error, vacío}

Primer paso: Asignación de tipo

P à D ; S
D à D ; D
D à id : T {Añadir_TS(id.entrada, T.tipo}
T à char {T.tipo = char}
T à entero {T.tipo = entero}
T à real {T.tipo = real}
T à booleano {T.tipo = booleano}
T à array[num] of T1 {T.tipo = array(num.val, T1.tipo)}
T à ^T1 {T.tipo = puntero(T1.tipo)}
T à T1 à T2 {T.tipo = T1.tipo à T2.tipo}

Segundo paso: Comprobación de tipo en expresiones

E à literal {E.tipo = char}

E à num {E.tipo = entero}

E à id {E.tipo = Consultar_TS(id.entrada)}

E à id[E1]

{id.tipo = Consultar_TS(id.entrada)}

{E.tipo =si (id.tipo=array(s,t) y E2.tipo=entero)
entonces t
sino error_tipo}

E à E1 op_lógico E2

{E.tipo = si (E1.tipo=booleano y E2.tipo=booleano)
entonces booleano
sino error_tipo}

E à id^

{id.tipo = Consultar_TS(id.entrada)}

{E.tipo = si (id.tipo = puntero(t))
entonces t
sino error_tipo}

E à E1 mod E2

{E.tipo = si (E1.tipo=entero y E2.tipo=entero)
entonces entero
sino error_tipo}

E à id (E2)

{id.tipo = Consultar_TS(id.entrada)}

{E.tipo = si id.tipo = s y E1.tipo= s à t)
entonces t
sino error_tipo}

Tercer paso: Comprobación de tipo en sentencias

S à id:=E	{id.tipo = Consultar_TS(id.entrada)}
	{S.tipo = si (id.tipo = E.tipo) entonces vacio sino error_tipo}
S à if E then S1	{S.tipo = si (E.tipo = booleano) entonces S1.tipo sino error_tipo}
S à while E do S1	{S.tipo = si (E.tipo = booleano) entonces S1.tipo sino error_tipo}
S à S1; S2	{S.tipo = si (S1.tipo=vacio y S2.tipo=vacio) entonces vacio sino error_tipo}