

Análisis Semántico (I)

Diseño de Lenguajes de Programación

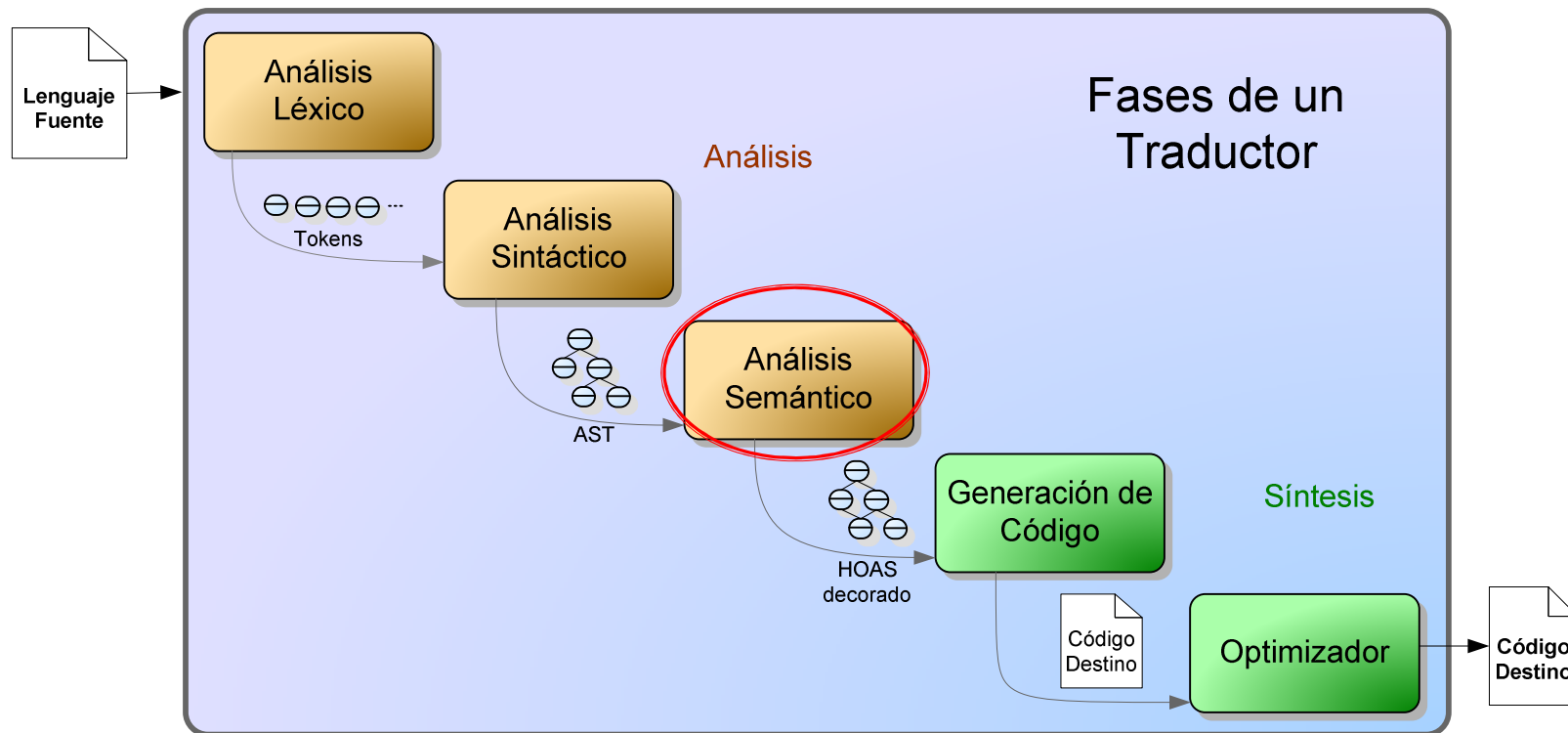
Ingeniería Informática

Universidad de Oviedo

(v1.14)

Raúl Izquierdo Castanedo

Usted está aquí...



¿Qué hace el Análisis Semántico?

¿Qué hace?

Sintaxis de un lenguaje

- Conjunto de reglas formales que especifican la estructura de sus programas

```
print a >> b;  
a = b;
```

Semántica de un lenguaje

- Conjunto de reglas que especifican el significado de cualquier sentencia sintácticamente válida

```
print a >> b;  
a = b;
```

Objetivo del Análisis Semántico o Contextual

- Comprobar la validez semántica de las sentencias aceptadas por el analizador sintáctico.

```
c = a + b;           // Léxica y sintácticamente válido, pero... ¿es válido?
```

Errores a detectar

Ejemplos de errores no detectados en fases anteriores (léxico y sintáctico)

- Chequeos de enlace
 - Uso de símbolos no definidos
- Chequeos de unicidad
 - Definiciones repetidas
 - Campos en una estructura
 - Enumerados
 - Sobrecarga de Funciones
- Chequeos de Tipo
 - Expresiones
 - Que los operadores se apliquen a operandos del tipo adecuado
 - Número y tipo de los argumentos
 - Asignaciones *compatibles*
- Chequeos de control de flujo
 - Cada rama de una función debe retornar un valor
 - No puede haber un break fuera de un switch, o bucle *while* o *do-while*
- ...

¿Por qué no se han detectado aún?

¿Por qué no se han detectado aún dichos errores?

- O no se puede con una gramática libre de contexto
 - Uso de una variable previamente declarada
 - Lenguaje $\{ a^n b^n c^n \}$
- O es mucho más difícil
 - *return*
 - *break* o *continue*

Una manera [muy] informal de describir lo que hay que hacer en el Análisis Semántico:

“Todo aquello que no se puede o es difícil de hacer con una gramática libre de contexto (tipo 2)”

¿Cuándo se hace el Análisis
Semántico?

¿Cuándo se hace?

¿Puede un compilador detectar todos los errores?

¿Cuándo se realiza el Análisis Semántico?

Tipos de Análisis Semántico por el momento de su realización

- Estático
- Dinámico

Ejemplo de lenguajes por el tipo de Análisis Semántico que realizan:

- Ambos
 - Java, C#
- Solo estático
 - C
- Solo dinámico
 - Javascript, Python, Ruby, Lisp, ...

En nuestro lenguaje se hará Análisis Semántico...

¿Cómo se hace el Análisis Semántico
[estático]?

¿Cómo se hace?

El analizador sintáctico ha dejado pasar un AST con combinaciones de nodos que no son válidas

5 = a;

8 && (b + 2) *// b es real!!*

El analizador semántico busca dichas combinaciones

- Comprueba que cada subárbol cumple determinadas condiciones
 - Si cada subárbol es válido, *el árbol completo es válido*

Necesidades

Por tanto, el Analizador Semántico necesita:

1. Poder añadir condiciones que deben cumplirse (predicados) a los nodos
 - ❑ Los dos operandos de un AND deben ser enteros
 - ❑ Lo que esté a la izquierda de una asignación no puede ser constante
2. Añadir a los nodos la información *adicional* (atributos) que necesitan los predicados
 - ❑ No suele ser suficiente con la que hay en el AST

$8 \ \&\& \ (b + 2)$

¿Cómo se especifican los requisitos anteriores?

- ❑ ¿Usamos el lenguaje natural?

Metalinguaje

Gramáticas Atribuidas (GAt)

Gramáticas Atribuidas

Notación (metalenguaje) para indicar

- Qué atributos se añaden a qué símbolos
- Cómo toman estos su valor

Se puede usar en cualquier fase que quiera añadir información a los símbolos y/o comprobar la estructura de éstos

- Semántico
- Gestión de memoria
- Generación de código (si éste se considera un atributo)
- Cálculo de valores (SÓLO si es un intérprete)
- ...

Definición de GAt

Gramática Atribuida (GAt)

- Es una extensión de una Gramática G

$$GAt = \{ G, A, R, B \}$$

Donde

- G es una Gramática. Puede ser Libre de Contexto (GLC) o Abstracta (GAb)
- $B = \{ \bigcup B(p) \} / B(p)$ son los predicados asociados a $p \in P$
 - Deben cumplirse **todos** para que la entrada sea válida
- $A = \{ \bigcup A(x) \} / A(x)$ son los atributos de $x \in (VT \cup VN)$
 - Puede representar cualquier información que se necesite en un nodo
 - Tipo de una expresión
 - Dirección de una variable
 - Valor de una expresión (NO SI ES un compilador)
 - Fuente y color para coloreado de sintaxis (*syntax highlighting*)
 - Cada atributo tiene asociado un dominio (int, string, Tipo, ...)
 - Si a es un atributo del símbolo x , entonces $x.a$ representa su valor
expr.tipo, definición.dirección, ...
- $R = \{ \bigcup R(p) \} / R(p)$ son las reglas semánticas asociadas a $p \in P$
 - Sea $p = X_0 \rightarrow X_1 X_2 \dots X_n \in P$
 - Una regla semántica asociada a p es de la forma
$$X_i.a_j = f(X_0.a_{1'}, \dots, X_0.a_{k'}, X_1.a_{1'}, \dots, X_1.a_{k'}, \dots, X_n.a_{1'}, \dots, X_n.a_{k'})$$

Ejemplo (I)

Queremos reconocer el siguiente lenguaje

$$\{ a^n b^n c^n \}$$

- Ejemplos

abc

aab**b**cc

aaab**bb**ccc

Problema

- Una gramática libre de contexto no es capaz de generar un lenguaje que genere *sólo* dichas sentencias

¿Solución con Gramáticas Atribuidas?

1. Usar una GLC **aunque** genere sentencias adicionales
2. Aumentar la GLC a Gramática Atribuida añadiéndole condiciones que rechacen las sentencias adicionales no válidas

Ejemplo (II)

Paso 1

- Hacer una Gramática que genere el lenguaje

Gramática Libre de Contexto

sentencia \rightarrow listaA listaB listaC

listaA \rightarrow A

listaA \rightarrow listaA A

listaB \rightarrow B

listaB \rightarrow listaB B

listaC \rightarrow C

listaC \rightarrow listaC C

Genera sentencias no válidas

abbccc

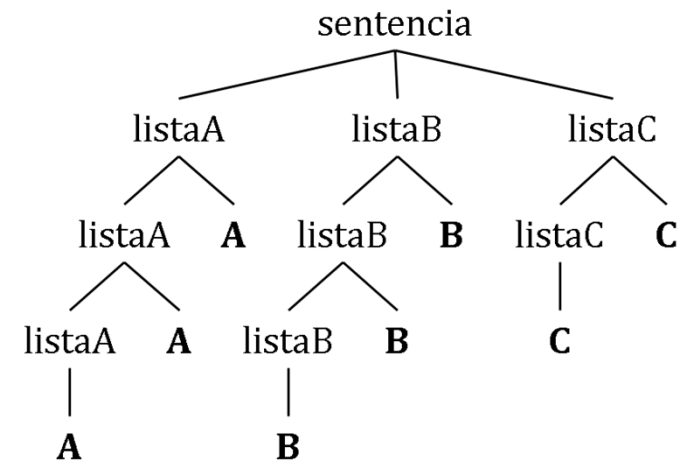
Ejemplo (III)

Cadena de entrada:
aaabbbcc

Paso 2. Extender la gramática

- Añadir predicados
- ... atributos...
- ... y reglas semánticas...

Símbolo	Atributos A(x)	Dominio
listaA	nivel	entero
listaB	nivel	entero
listaC	nivel	entero

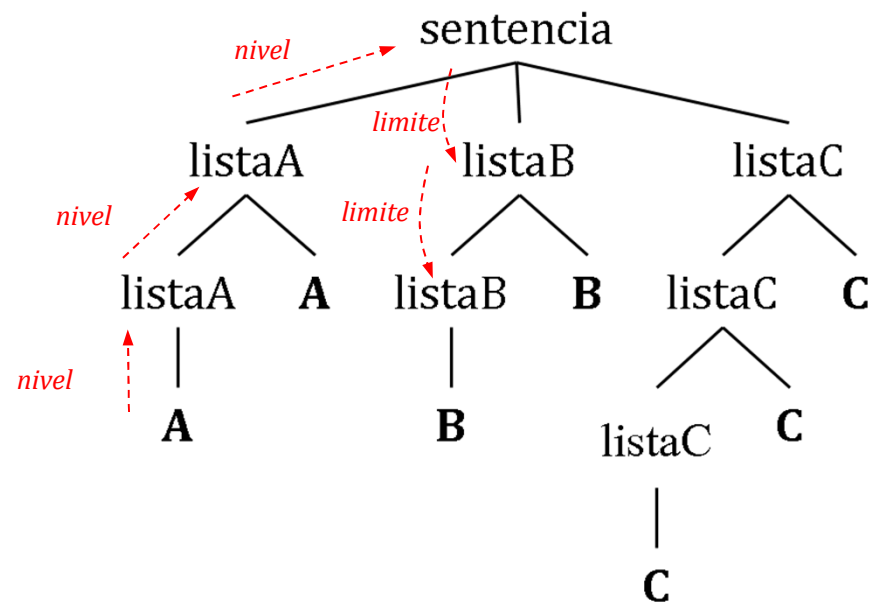


Gramática Libre de Contexto	Predicados B(p)	Reglas semánticas R(p)
sentencia \rightarrow listaA listaB listaC	listaA.nivel == listaB.nivel listaB.nivel == listaC.nivel	
listaA \rightarrow A		listaA.nivel = 1
listaA ₀ \rightarrow listaA ₁ A		listaA ₀ .nivel = listaA ₁ .nivel + 1
listaB \rightarrow B		listaB.nivel = 1
listaB ₀ \rightarrow listaB ₁ B		listaB ₀ .nivel = listaB ₁ .nivel + 1
listaC \rightarrow C		listaC.nivel = 1
listaC ₀ \rightarrow listaC ₁ C		listaC ₀ .nivel = listaC ₁ .nivel + 1

Gramática Libre de Contexto	Predicados B(p)	Reglas semánticas R(p)
sentencia \rightarrow listaA listaB listaC		listaB.limite = listaA.nivel listaC.limite = listaA.nivel
listaA \rightarrow A		listaA.nivel = 1
listaA ₀ \rightarrow listaA ₁ A		listaA ₀ .nivel = listaA ₁ .nivel + 1
listaB \rightarrow B	listaB.limite == 1	
listaB ₀ \rightarrow listaB ₁ B		listaB ₁ .limite = listaB ₀ .limite - 1
listaC \rightarrow C	listaC.limite == 1	
listaC ₀ \rightarrow listaC ₁ C		listaC ₁ .limite = listaC ₀ .limite - 1

Entrada
aabbccc

Símbolo	Atributos	Dominio
listaA	nivel	entero
listaB	limite	entero
listaC	limite	entero



Tipos de Atributos (I)

En una regla semántica asociada a $p \in P$ se puede asignar valor a un atributo de...

- a) ... el símbolo antecedente de la regla (padre)
 - Entonces el atributo se llama **Sintetizado**
 - Se usan cuando se quiere pasar información de los hijos al padre
- b) ... un símbolo del consecuente de la regla (hijo)
 - Entonces el atributo se llama **Heredado**
 - Se usan cuando se quiere pasar información del padre a los hijos

¿De qué tipos son t1, t2 y t3?

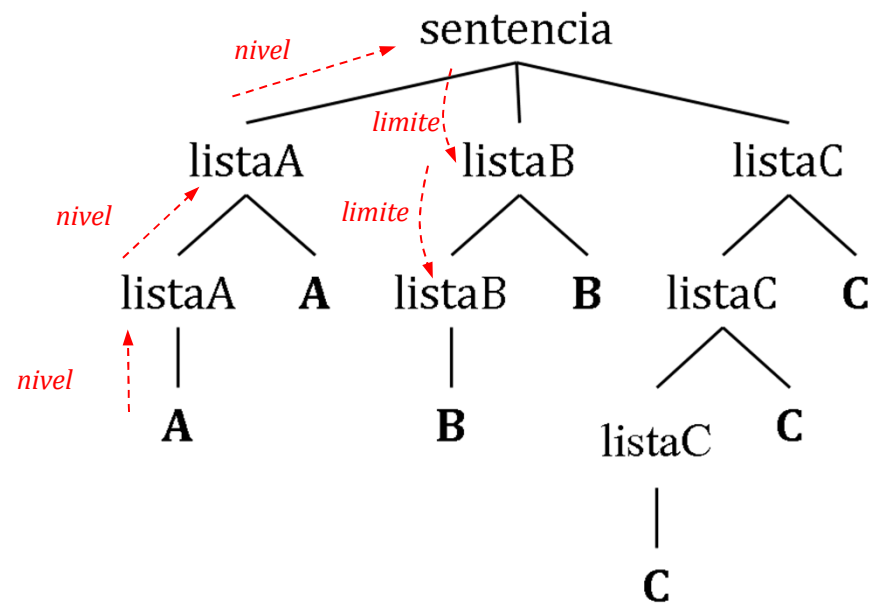
GLC	Predicados B(p)	Reglas semánticas R(p)
$s \rightarrow \dots$		
$a \rightarrow b c$		a.t1 = ... b.t2 = ...
$b \rightarrow IDENT$		b.t3 = ...
$c \rightarrow \dots$		

Símbolo	Atributo A(x)	H/S
a	t1	i?
b	t2	i?
b	t3	i?

Gramática Libre de Contexto	Predicados B(p)	Reglas semánticas R(p)
sentencia \rightarrow listaA listaB listaC		listaB.limite = listaA.nivel listaC.limite = listaA.nivel
listaA \rightarrow A		listaA.nivel = 1
listaA ₀ \rightarrow listaA ₁ A		listaA ₀ .nivel = listaA ₁ .nivel + 1
listaB \rightarrow B	listaB.limite == 1	
listaB ₀ \rightarrow listaB ₁ B		listaB ₁ .limite = listaB ₀ .limite - 1
listaC \rightarrow C	listaC.limite == 1	
listaC ₀ \rightarrow listaC ₁ C		listaC ₁ .limite = listaC ₀ .limite - 1

¿De qué tipo es cada atributo?

Símbolo	Atributos	Dominio	H/S
listaA	nivel	entero	?
listaB	limite	entero	?
listaC	limite	entero	?



Tipos de Atributos (II)

Dada una regla $p \in P$

$$X_0 \rightarrow X_1 X_2 \dots X_n$$

- Una regla semántica asociada a p es de la forma

$$X_i.a_j = f(X_0.a_1, \dots, X_0.a_k, X_1.a_1, \dots, X_1.a_k, \dots, X_n.a_1, \dots, X_n.a_k)$$

Se dice que un atributo a de un símbolo X es

- Sintetizado si la regla que lo asigna tiene la forma

$$X_0.a = f(X_1.a_1, \dots, X_1.a_k, \dots, X_n.a_1, \dots, X_n.a_k)$$

- El valor del atributo se calcula a partir de los atributos de los hijos del símbolo

- Heredado si la regla que lo asigna tiene la forma

$$X_i.a = f(X_0.a_1, \dots, X_0.a_k, X_1.a_1, \dots, X_1.a_k, \dots, X_n.a_1, \dots, X_n.a_k), \quad i \neq 0$$

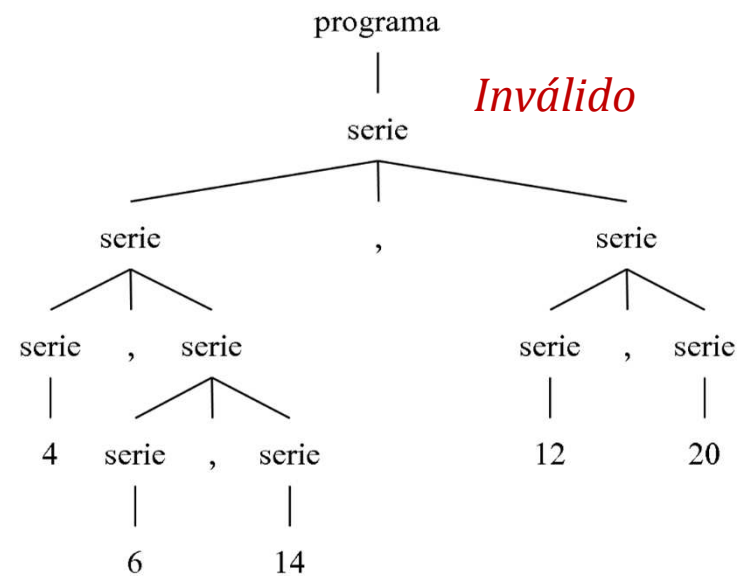
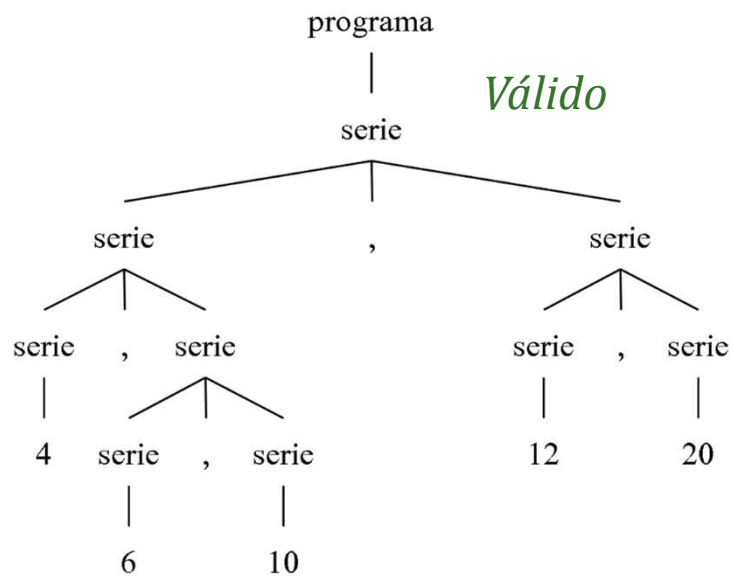
- El valor del atributo se calcula a partir de los atributos de los hermanos y el padre del símbolo

Ejercicio E1

Sea la siguiente gramática

Símbolo	Atributos A(x)	Dominio	Tipo

Gramática Libre de Contexto	Predicados B(p)	Reglas semánticas R(p)
programa \rightarrow serie		
serie \rightarrow LITENT		
serie ₀ \rightarrow serie ₁ ; serie ₂		



Implementación de Gramáticas Atribuidas

Implementación de Gramáticas Atribuidas

Para implementar una GAt hay que determinar

- Cómo se implementan los *atributos* de los símbolos
- En qué orden se evalúan las *reglas semánticas y predicados*


Implementación de Atributos

Implementación de Atributos (I)

Un **atributo** de un **símbolo** de una Gramática Atribuida se puede implementar:


Símbolo	Atributos A(x)	Dominio
definicionDeVariable	direccion	entero

Como un **atributo** de una clase



```
public class DefinicionDeVariable ...
{
    public int direccion;
    ...
}
```

Como una **propiedad** de una clase



```
public class DefinicionDeVariable ...
{
    private int direccion;

    int getDireccion() {
        return direccion;
    }

    public void setDireccion(int direccion) {
        this.direccion = direccion;
    }

    ...
}
```

Implementación de Atributos (II)

¿Cómo se implementa como *propiedad* cuando el atributo es de una *categoría sintáctica*?

- NO se repite el código en cada clase!!!

```
interface Expresion extends AST {  
    int getX();  
    void setX(int x);  
}
```

```
abstract class AbstractExpresion implements Expresion {  
    private int x;  
  
    public int getX() {  
        return x;  
    }  
  
    public void setX(int x) {  
        this.x = x;  
    }  
}
```

```
class LiteralEntero extends AbstractExpresion { ... }  
class Variable extends AbstractExpresion { ... }  
class Suma extends AbstractExpresion { ... }
```

Símbolo	Atributo	Dominio
expr	x	int

Gramática Abstracta (GAb)
...
literalEntero:expr →
variable:expr →
suma:expr → left:expr right:expr
...

Implementación de Reglas Semánticas y Predicados

Patrón Visitor. Repaso [*mu*y] rápido

Nodos del AST

```
public interface AST {  
    void accept(Visitor v);  
}
```

Redefiniendo el método *accept* se elige el *visit* correspondiente al nodo

```
class Print implements Sentencia {  
    Expression expr;  
  
    void accept(Visitor v) {  
        v.visitPrint(this);  
    }  
}
```

```
class Suma implements Expression {  
    Expression left, right;  
  
    void accept(Visitor v) {  
        v.visitSuma(this);  
    }  
}
```

```
public class EjemploVisitor implements Visitor  
{  
    public Object visitProg(Programa prog) {  
        for (Sentencias sent : prog.sentencias)  
            sent.accept(this);  
  
        return null;  
    }  
  
    public Object visitPrint(Print print) {  
        print.expr.accept(this);  
  
        return null;  
    }  
  
    public Object visitSuma(Suma suma) {  
        suma.left.accept(this);  
        suma.right.accept(this);  
  
        return null;  
    }  
    ...  
}
```


Tareas a realizar en cada Nodo

¿Qué hay que hacer en el método *visit* de *cada* nodo?

- Lo que indique la GAt para dicho nodo
 - Hace de *pseudocódigo* del recorrido

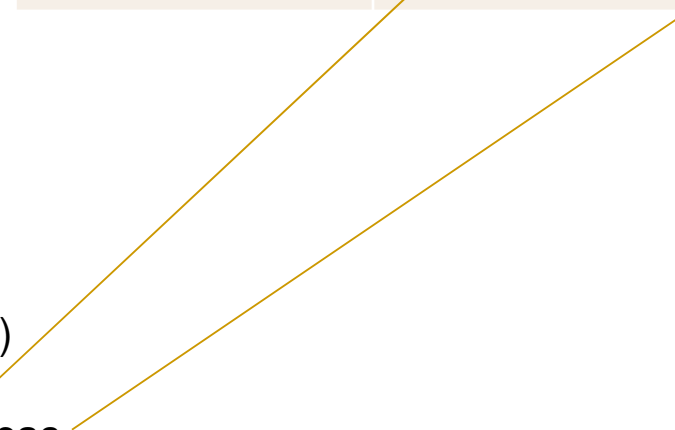
- Tareas a realizar

```
public Object visitSuma(Suma suma)
{
    ???
    suma.left.accept(this);
    ???
    suma.right.accept(this);
    ???
}
```



- Recorrer los hijos (¿orden?)
- Comprobar los predicados
- Evaluar sus reglas semánticas
 - Asignar valor a sus atributos sintetizados
 - Asignar valor a los atributos heredados de sus hijos

Gramática Abstracta	Predicados B(p)	Reglas semánticas
""		
suma → left:expr right:expr	right.terminal == TRUE	suma.terminal = FALSE suma.prioridad = 1



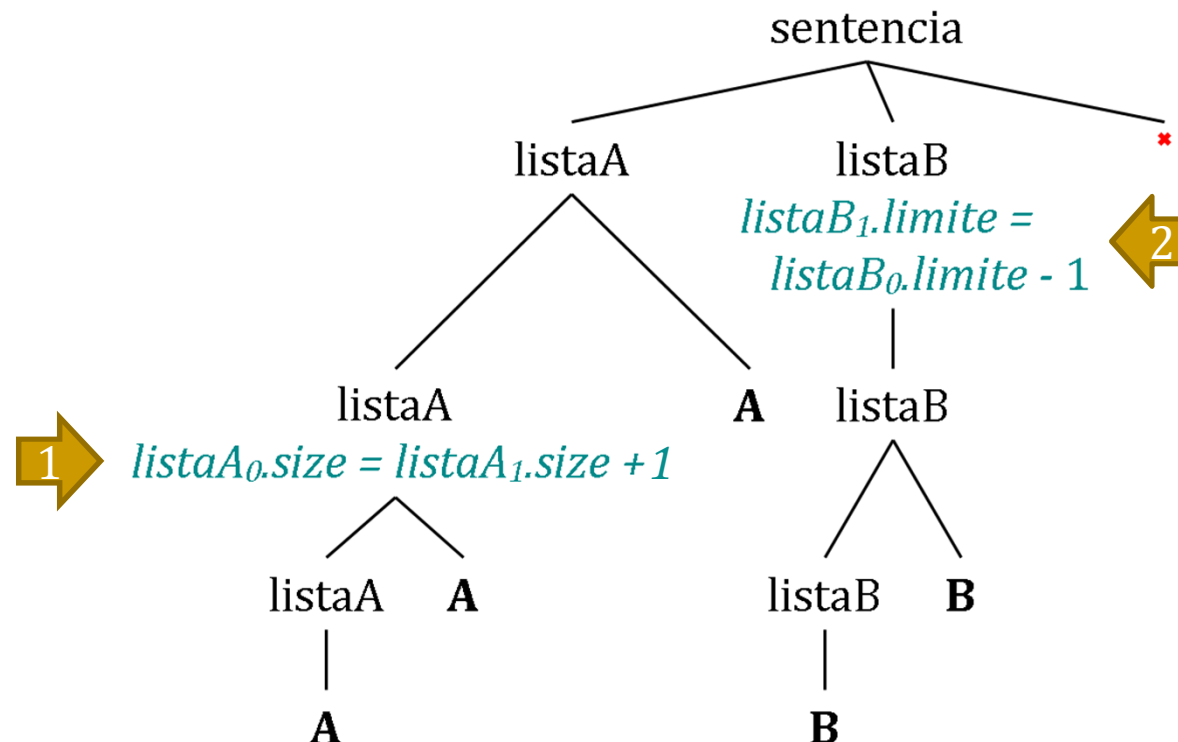
Problema ¿En qué orden hay que hacerlo?

- ¿En qué orden se visitan los hijos?
- ¿Se asigna valor a los atributos antes o después de visitar los hijos?

Orden de Recorrido (II)

Las Gramáticas Atribuidas no establecen un orden

- Son declarativas
 - Pero el orden es importante!!!



Orden de Recorrido (III)

Por tanto, el problema es...

- Que en cada método visit hay que establecer un orden de evaluación de las reglas y de visitas a los hijos
 - Garantizando que *a la hora de usar un atributo ya haya sido inicializado*
 - Es decir, que se hayan recorrido ya aquellos nodos que realizan la asignación de dichos atributos

Un recorrido que cumpla lo anterior se le llama *orden topológico*

- Y, dada una GAt, ¿cómo nos aseguramos de implementar un orden topológico?
 - Depende del tipo de Gramática Atribuida

Tipos de Gramáticas Atribuidas

Tipos de Gramáticas Atribuidas

Las GAt se clasifican en función del tipo de reglas semánticas que tengan

- Para simplificar, a partir de ahora se incluirán los predicados cuando se hable de reglas semánticas

En la práctica, la mayoría de las Gramáticas Atribuidas usadas en la construcción de traductores son

- S-Atribuidas
- L-Atribuidas
 - Para las cuales un orden topológico está ya predefinido

Gramáticas S-Atribuidas

Una Gramática Atribuida es S-Atribuida si todos sus atributos son *sintetizados*

- Si toda regla semántica es de la forma

$$X_0.a = f(X_1.a_1, \dots, X_1.a_k, \dots, X_n.a_k)$$

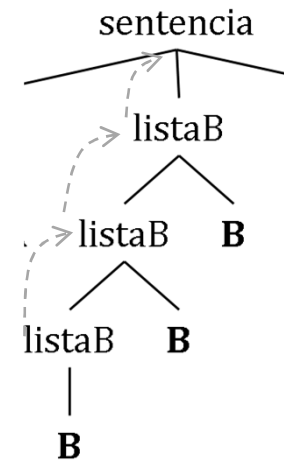
donde

$$X_0 \rightarrow X_1 X_2 \dots X_n$$

- Es decir, si ningún nodo requiere que su padre o hermano le pase un valor

GLC	B(p)	Reglas semánticas R(p)
$\text{listaA}_0 \rightarrow \text{listaA}_1 A$		$\text{listaA}_0.\text{tamaño} = \text{listaA}_1.\text{tamaño} + 1$
$\text{listaA} \rightarrow A$		$\text{listaA}.\text{tamaño} = 1$

- La información fluye siempre hacia arriba



Recorrido de Gramáticas S-Atribuidas

Características de una Gramática S-Atribuida

- Puede implementarse con un solo recorrido del árbol
 - Es un clásico recorrido en *postorden*

La implementación de *todo* método *visit* debe seguir el siguiente patrón:

Sea p una regla con la forma " $padre \rightarrow hijo_1 \dots hijo_n$ "

```
visit(Nodo padre) {  
  Para cada  $hijo_i$  { // Da igual el orden de visita de los hijos  
    visitar( $hijo_i$ ) // Al volver, el hijo tendrá ya asignados todos sus atributos  
  }  
  Comprobar los predicados asociados a la regla  $p$  ( $B(p)$ )  
  Asignar valor a los atributos de padre ( $R(p)$ )  
}
```

- El patrón anterior, nos asegura un *orden topológico*

Ejemplo (I)

Implementar la siguiente Gramática Atribuida

Símbolo	Atributos A(x)	Dominio	Tipo	Gramática Abstracta	Predicados B(p)	Reglas semánticas R(p)
expr	terminal	boolean	?	programa → expr		
expr	prioridad	entero	?	litEnt → lexema:string		litEnt.terminal = TRUE litEnt.prioridad = 3
				variable → lexema:string		variable.terminal = TRUE variable.prioridad = 3
				suma → left:expr right:expr	right.terminal == TRUE	suma.terminal = FALSE suma.prioridad = 1
				mult → left:expr right:expr	right.terminal == TRUE left.prioridad ≥ 2	mult.terminal = FALSE mult.prioridad = 2

La implementación de los nodos es la siguiente:

```
class Programa { Expr expr; }  
interface Expr extends AST { }  
  
class LitEnt implements Expr { String lexema; }  
class Variable implements Expr { String lexema; }  
class Suma implements Expr { Expr left; Expr right; }  
class Multiplicacion implements Expr { Expr left; Expr right; }
```

Ejemplo (II)

Paso 1. Añadir atributos

- Diseño de clases inicial

```
class Programa { Expr expr; }  
interface Expr extends AST { }
```

```
class LitEnt implements Expr { String lexema; }  
class Variable implements Expr { String lexema; }  
class Suma implements Expr { Expr left; Expr right; }  
class Multiplicacion implements Expr { Expr left; Expr right; }
```

- Atributos a añadir

Símbolo	Atributos A(x)	Dominio	Tipo
expr	terminal	boolean	sintetizado
expr	prioridad	entero	sintetizado

- Cambios en el diseño parar añadir los atributos

```
interface Expr extends AST {  
    void setTerminal(boolean terminal);  
    boolean getTerminal();  
  
    void setPrioridad(int prioridad);  
    int getPrioridad();  
}
```

```
abstract class AbstractExpr implements Expr {  
    private boolean terminal;  
    private int prioridad;  
  
    public void setTerminal(boolean terminal) { this.terminal = terminal; }  
    public boolean getTerminal() { return terminal; }  
  
    public void setPrioridad(int prioridad) { this.prioridad = prioridad; }  
    public int getPrioridad() { return prioridad; }  
}
```

```
class LitEnt extends AbstractExpr { String lexema; }  
class Variable extends AbstractExpr { String lexema; }  
class Suma extends AbstractExpr { Expr left; Expr right; }  
class Multiplicacion extends AbstractExpr { Expr left; Expr right; }
```

Ejemplo (III)

Paso 2

- Implementar reglas semánticas y predicados

Gramática Abstracta	Predicados B(p)	Reglas semánticas R(p)
programa → expr		
litEnt → lexema:string		litEnt.terminal = TRUE litEnt.prioridad = 3
variable → lexema:string		variable.terminal = TRUE variable.prioridad = 3
suma → left:expr right:expr	right.terminal == TRUE	suma.terminal = FALSE suma.prioridad = 1
mult → left:expr right:expr	right.terminal == TRUE left.prioridad ≥ 2	mult.terminal = FALSE mult.prioridad = 2

```
visit(Nodo padre) {  
  Para cada hijoi { // Da igual el orden de visita de los hijos  
    visitar(hijoi) // Al volver, el hijo tendrá ya asignados todos sus atributos  
  }  
  Comprobar los predicados asociados a la regla p (B(p))  
  Asignar valor a los atributos de padre (R(p))  
}
```

```
public class GramáticaAtribuida implements Visitor {
```

```
  public Object visit(Programa prog, Object param) {  
    prog.expr.accept(this, param);  
    return null;  
  }
```

```
  public Object visit(LitEnt litEnt, Object param) {  
    litEnt.setTerminal(true);  
    litEnt.setPrioridad(3);  
    return null;  
  }
```

```
  public Object visit(Variable var, Object param) {  
    var.setTerminal(true);  
    var.setPrioridad(3);  
    return null;  
  }
```


Ejemplo (IV)

GAb	Predicados B(p)	Reglas semánticas R(p)
programa → expr		
litEnt → string		litEnt.terminal = TRUE litEnt.prioridad = 3
variable → string		variable.terminal = TRUE variable.prioridad = 3
suma → left:expr right:expr	right.terminal == TRUE	suma.terminal = FALSE suma.prioridad = 1
mult → left:expr right:expr	right.terminal == TRUE left.prioridad ≥ 2	mult.terminal = FALSE mult.prioridad = 2

```

public Object visit(Suma suma, Object param) {
    suma.left.accept(this, param);
    suma.right.accept(this, param);

    predicado(suma.right.getTerminal() == true);

    suma.setTerminal(false);
    suma.setPrioridad(1);

    return null;
}

```

```

public Object visit(Multiplicacion mult, Object param) {
    mult.left.accept(this, param);
    mult.right.accept(this, param);

    predicado(mult.right.getTerminal());
    predicado(mult.left.getPrioridad() >= 2);

    mult.setTerminal(false);
    mult.setPrioridad(2);
    return null;
}

```

```

private void predicado(boolean condición) {
    if (!condición)
        System.out.println("Error semántico");
}

```

```

visit(Nodo padre) {
    Para cada hijoi { // Da igual el orden de visita de los hijos
        visitar(hijoi) // Al volver, el hijo tendrá ya asignados todos sus atributos
    }
    Comprobar los predicados asociados a la regla p (B(p))
    Asignar valor a los atributos de padre (R(p))
}

```

Gramáticas L-Atribuidas

Una Gramática atribuida es L-Atribuida si

Para toda producción p de la forma

$$X_0 \rightarrow X_1 X_2 \dots X_n$$

todos los atributos de X_i ($1 \leq i \leq n$) son calculados a partir de

- Los atributos de X_0 (heredados)
- Los atributos de sus hermanos a su izquierda

$$X_i.a = f(X_0.a_1, \dots, X_0.a_k, X_1.a_1, \dots, X_1.a_k, \dots, X_{i-1}.a_1, \dots, X_{i-1}.a_k)$$

- Dicho de otra forma
 - Una GAt no puede ser L-Atribuida si existe algún símbolo que tenga un atributo cuyo valor dependa de un hermano de su derecha

GLC	Predicados	R(p)
sentencia \rightarrow listaA listaB listaC		listaB.limite = listaA.tamaño listaC.limite = listaA.tamaño
...		
listaB ₀ \rightarrow listaB ₁ B		listaB ₁ .limite = listaB ₀ .limite - 1

¿Ejemplo de regla que impidiera ser L-Atribuida?

- Ojo. No confundir
 - Una gramática S-Atribuida es aquella que todos sus atributos son sintetizados y una gramática L-Atribuida es aquella que además tiene heredados **FALSO**
- Toda gramática S-Atribuida es también L-Atribuida
 - ¿Por qué?

Recorrido de Gramáticas L-Atribuidas

Características de una Gramática L-Atribuida

- Puede implementarse con un solo recorrido del árbol

La implementación de todo *visit* debe seguir el siguiente patrón:

Sea p una regla con la forma " $padre \rightarrow hijo_1 \dots hijo_n$ "

```
visit(Nodo padre) {  
  Para cada  $hijo_i$  { // Los hijos deben recorrerse de izquierda a derecha  
    Asignar valor a los atributos de  $hijo_i$  que sean heredados  
    visitar( $hijo_i$ ) // Al volver, el hijo tendrá ya asignados todos sus atributos sintetizados  
  }  
  Comprobar los predicados asociados a la regla  $p$  ( $B(p)$ )  
  Asignar valor a los atributos sintetizados de padre ( $R(p)$ )  
}
```

Comparar con...

```
visit(Nodo padre) {  
  Para cada  $hijo_i$  { // Da igual el orden de visita de los hijos  
    visitar( $hijo_i$ ) // Al volver, el hijo tendrá ya asignados todos sus atributos  
  }  
  Comprobar los predicados asociados a la regla  $p$  ( $B(p)$ )  
  Asignar valor a los atributos de padre ( $R(p)$ )  
}
```

Recorrido de otras Gramáticas Atribuidas

¿Cómo determinar el orden de evaluación de una gramática que no sea de los dos tipos anteriores?

- Opción 1. Análisis *manual* de la gramática
 - Se debe decidir
 - En qué orden hay que recorrer los hijos
 - Como intercalar la evaluación de las reglas semánticas entre ellos
 - Cuántas veces hay que recorrer cada hijo
 - ¿Cuándo habría que recorrer varias veces un mismo hijo?
 - Suele ser sencillo pero no se tiene garantía de haber obtenido un orden topológico
- Opción 2. Dividir la Gramática Atribuida
 - Se divide en dos o más cada una de las cuales sea un caso de las dos anteriores
 - Más sencillo de implementar por separado
 - Se vuelve a garantizar un orden topológico
 - De hecho así se implementará el analizador semántico
 - Aunque se podría hacer en una GAt, se separará en
 - Una GAt para la Fase de Identificación
 - Otra GAt para la Fase de Comprobación de Tipos
 - Cada una de las cuales es S/L-Atribuida

Ejercicio E2 (I). Implementación de GAt

Implementar la siguiente Gramática Atribuida

Gramática Abstracta	Predicados B(p)	Reglas semánticas R(p)
programa → expr		expr.terminal = FALSE
litEnt → lexema:string		litEnt.prioridad = 3
variable → lexema:string		variable.prioridad = 3
suma → left:expr right:expr	suma.terminal == FALSE	left.terminal = FALSE right.terminal = TRUE suma.prioridad = 1
mult → left:expr right:expr	mult.terminal == FALSE left.prioridad ≥ 2	left.terminal = FALSE right.terminal = TRUE mult.prioridad = 2

Símbolo	Atributos A(x)	Dominio	Tipo
expr	terminal	boolean	¿?
expr	prioridad	entero	¿?

La implementación de los nodos es la siguiente:

```
class Programa { Expr expr; }  
interface Expr extends AST { }
```

```
class LitEnt implements Expr { String lexema; }  
class Variable implements Expr { String lexema; }  
class Suma implements Expr { Expr left; Expr right; }  
class Multiplicacion implements Expr { Expr left; Expr right; }
```

Ejercicio E2 (II)

```
public class GramáticaAtribuida implements Visitor {
```

```
    private void predicado(boolean condición) {  
        if (!condición)  
            System.out.println("Error semántico");  
    }
```

```
    public Object visit(Programa prog, Object param) {
```

```
        prog.expr.accept(this, param);
```

```
        return null;  
    }
```

```
    public Object visit(LitEnt litEnt, Object param) {
```

```
        return null;  
    }
```

```
    public Object visit(Variable var, Object param) {
```

```
        return null;  
    }
```

```
    public Object visit(Suma suma, Object param) {
```

```
        suma.left.accept(this, param);
```

```
        suma.right.accept(this, param);
```

```
        return null;  
    }
```

```
    public Object visit(Multiplicacion mult, Object param) {
```

```
        mult.left.accept(this, param);
```

```
        mult.right.accept(this, param);
```

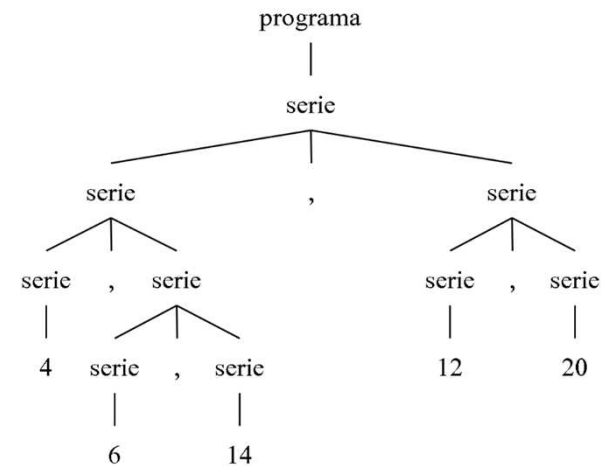
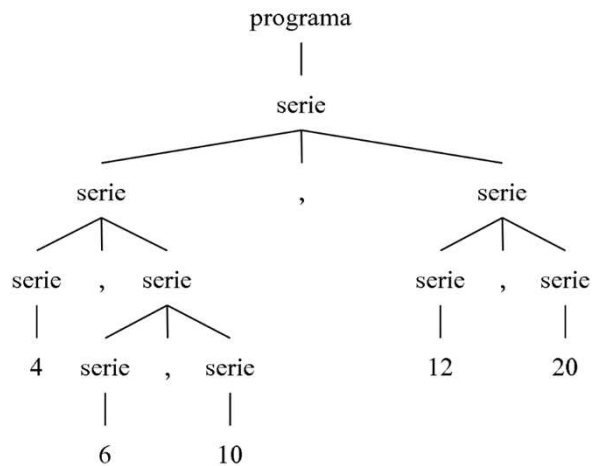
```
        return null;  
    }
```

Soluciones

Solución E1

Símbolo	Atributos A(x)	Dominio	Tipo
serie	mayor	entero	sintetizado
serie	menor	entero	sintetizado

Gramática Libre de Contexto	Predicados B(p)	Reglas semánticas R(p)
programa \rightarrow serie		
serie \rightarrow LITENT		serie.mayor = LITENT.lexema serie.menor = LITENT.lexema
serie ₀ \rightarrow serie ₁ , serie ₂	serie ₁ .mayor < serie ₂ .menor	serie ₀ .mayor = serie ₂ .mayor serie ₀ .menor = serie ₁ .menor



Ejemplo (II)

Paso 1. Añadir atributos

- Diseño de clases inicial

```
class Programa { Expr expr; }  
interface Expr extends AST { }
```

```
class LitEnt implements Expr { String lexema; }  
class Variable implements Expr { String lexema; }  
class Suma implements Expr { Expr left; Expr right; }  
class Multiplicacion implements Expr { Expr left; Expr right; }
```

- Atributos a añadir

Símbolo	Atributos A(x)	Dominio	Tipo
expr	terminal	boolean	sintetizado
expr	prioridad	entero	sintetizado

Idéntico a lo anterior

- Cambios en el diseño para añadir los atributos

```
interface Expr extends AST {  
    void setTerminal(boolean terminal);  
    boolean getTerminal();  
    void setPrioridad(int prioridad);  
    int getPrioridad();  
}
```

```
abstract class AbstractExpr implements Expr {  
    private boolean terminal;  
    private int prioridad;  
  
    public void setTerminal(boolean terminal) { this.terminal = terminal; }  
    public boolean getTerminal() { return terminal; }  
  
    public void setPrioridad(int prioridad) { this.prioridad = prioridad; }  
    public int getPrioridad() { return prioridad; }  
}
```

```
class LitEnt extends AbstractExpr { String lexema; }  
class Variable extends AbstractExpr { String lexema; }  
class Suma extends AbstractExpr { Expr left; Expr right; }  
class Multiplicacion extends AbstractExpr { Expr left; Expr right; }
```

Solución E2 (II)

Paso 2

- Implementar reglas semánticas y predicados

Gramática Abstracta	Predicados B(p)	Reglas semánticas R(p)
programa → expr		expr.terminal = FALSE
litEnt → lexema:string		litEnt.prioridad = 3
variable → lexema:string		variable.prioridad = 3
suma → left:expr right:expr	suma.terminal == FALSE	left.terminal = FALSE right.terminal = TRUE suma.prioridad = 1
mult → left:expr right:expr	mult.terminal == FALSE left.prioridad ≥ 2	left.terminal = FALSE right.terminal = TRUE mult.prioridad = 2

```
visit(Nodo padre) {  
  Para cada hijoi { // Los hijos deben recorrerse de izquierda a derecha  
    Asignar valor a los atributos de hijoi que sean heredados  
    visitar(hijoi) // Al volver, el hijo tendrá ya asignados todos sus atributos sintetizados  
  }  
  Comprobar los predicados asociados a la regla p (B(p))  
  Asignar valor a los atributos sintetizados de padre (R(p))  
}
```

```
public class GramáticaAtribuida implements Visitor {
```

```
  private void predicado(boolean condición) {  
    if (!condición)  
      System.out.println("Error semántico");  
  }
```

```
  public Object visit(Programa prog, Object param) {  
    prog.expr.setTerminal(false);  
    prog.expr.accept(this, param);  
    return null;  
  }
```

```
  public Object visit(LitEnt litEnt, Object param) {  
    litEnt.setPrioridad(3);  
    return null;  
  }
```

```
  public Object visit(Variable var, Object param) {  
    var.setPrioridad(3);  
    return null;  
  }
```

Solución E2 (III)

Gramática Abstracta	Predicados B(p)	Reglas R(p)
programa → expr		expr.terminal = FALSE
litEnt → string		litEnt.prioridad = 3
variable → string		variable.prioridad = 3
suma → left:expr right:expr	suma.terminal == FALSE	left.terminal = FALSE right.terminal = TRUE suma.prioridad = 1
mult → left:expr right:expr	mult.terminal == FALSE left.prioridad ≥ 2	left.terminal = FALSE right.terminal = TRUE mult.prioridad = 2

```

visit(Nodo padre) {
  Para cada hijoi { // De izquierda a derecha
    Asignar valor a los atributos de hijoi que sean heredados
    visitar(hijoi)
  }
  Comprobar los predicados asociados a la regla p (B(p))
  Asignar valor a los atributos sintetizados de padre (R(p))
}

```

```

public Object visit(Suma suma, Object param) {
    suma.left.setTerminal(false);
    suma.left.accept(this, param);

    suma.right.setTerminal(true);
    suma.right.accept(this, param);

    predicado(suma.getTerminal() == false);

    suma.setPrioridad(1);

    return null;
}

```

```

public Object visit(Multiplicacion mult, Object param) {
    mult.left.setTerminal(false);
    mult.left.accept(this, param);

    mult.right.setTerminal(true);
    mult.right.accept(this, param);

    predicado(mult.getTerminal() == false);
    predicado(mult.left.getPrioridad() >= 2);

    mult.setPrioridad(2);

    return null;
}

```