
Análisis Semántico (II)

Diseño de Lenguajes de Programación
Ingeniería Informática
Universidad de Oviedo
(v2.1)

Raúl Izquierdo Castanedo

Notación Extendida de Gramáticas Atribuidas

Notación extendida de G-Atribuidas (I)

Reglas Semánticas

- En la forma *canónica* de las Gramáticas Atribuidas las reglas semánticas solo pueden ser asignaciones

$$X_i.a_j = f(X_0.a_1, \dots, X_0.a_k, X_1.a_1, \dots, X_1.a_k, \dots, X_n.a_1, \dots, X_n.a_k) \\ \text{con } X_0 \rightarrow X_1 X_2 \dots X_n \in P$$

Regla	Predicados B(p)	Reglas semánticas R(p)
sentencia \rightarrow listaA listaB listaC	...	
...		
listaA ₀ \rightarrow listaA ₁ A		listaA ₀ .size = listaA ₁ .size + 1
listaA \rightarrow A		listaA.size = 1

- De hecho, ni siquiera los predicados forman parte de la definición original de las Gramáticas Atribuidas
- Esta limitación las hace insuficientes en la práctica
 - Por tanto se hace necesario añadir extensiones a la notación de las reglas
 - Sobre las cuales no hay un consenso

Notación extendida de G-Atribuidas(II)

Notación que vamos a utilizar

■ Pseudocódigo *si/sino*

suma \rightarrow expr ₁ expr ₂	...	si expr ₁ .tipo == 'I' AND expr ₂ .tipo == 'I' suma.tipo = 'I' si no suma.tipo = 'R'
--	-----	---

```
mayor(tipoa, tipob) {  
  si (tipoa == 'I' AND tipob == 'I')  
    mayor = 'I'  
  si no  
    mayor = 'R'  
}
```

■ Funciones auxiliares (definidas aparte con esta notación)

suma \rightarrow expr ₁ expr ₂	...	suma.tipo = mayor (expr ₁ .tipo, expr ₂ .tipo)
--	-----	---

■ Estructuras de datos auxiliares (conjuntos, tablas hash, ...)

print \rightarrow expr	expr.tipo \in { int, real, char }	...
print \rightarrow expr	expr.tipo \in <i>TiposSimples</i>	...
print \rightarrow expr	{ expr.tipo } \cap <i>TiposSimples</i> $\neq \emptyset$...

<i>TiposSimples</i> = { int, real, char }

■ Operadores de conjuntos

$\in \notin \cup \cap \subset \not\subset$

■ Otros operadores

expr₁.tipo *is* TipoArray

expr₁ \neq null \Rightarrow esSimple(expr₁.tipo)

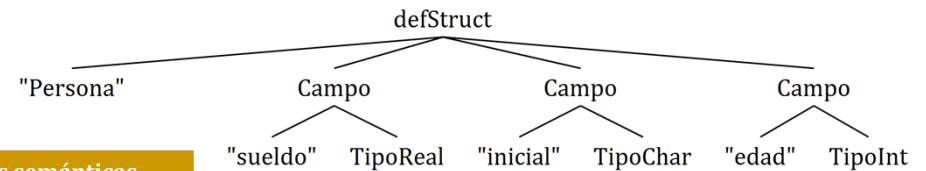
expr₁ == null \Leftrightarrow expr₂ == null

Notación extendida de G-Atribuidas(III)

Notación para Gramáticas Abstractas

- Notación para atributos multievaluados

Regla	Predicados	Reglas semánticas
...		
$\text{defStruct} \rightarrow \text{nombre:string } \text{campo}^*$		
$\text{campo} \rightarrow \text{nombre:string } \text{tipo}$		



En las reglas semánticas de *defStruct*, se podrán usar las siguientes expresiones:

$ \text{campo} $	Número de hijos (número de campos de la estructura)
campo_0	Primer hijo
campo_n	Último hijo (abreviatura de $\text{campo}_{ \text{campo} -1}$)
$\text{campo}[\text{<condición>}]$	El hijo que cumpla la condición (abreviatura de $\text{campo}_i / \text{<condición>} == \text{true}$) $\text{campo}[\text{nombre} == \text{"edad"}]$ (\emptyset si no existe)
campo_i	Para todo hijo (abreviatura de $\text{campo}_i / 0 \leq i \leq n$) $\text{campo}_i.\text{tipo} \in \{ \text{int}, \text{real}, \text{char} \}$ $\{ \text{campo}_i.\text{nombre} \}$ $\sum \text{campo}_i.\text{tipo.size}$

Criterio general para la notación a usar en las reglas semánticas

- Que sea lo más *precisa* posible...
 - ... y que facilite la posterior implementación de la GAt
 - Usar Java en caso de no encontrar adecuado nada de lo anterior

```

{
  <Sentencias Java entre llaves>
}
  
```

Ejemplo

Sea el siguiente lenguaje

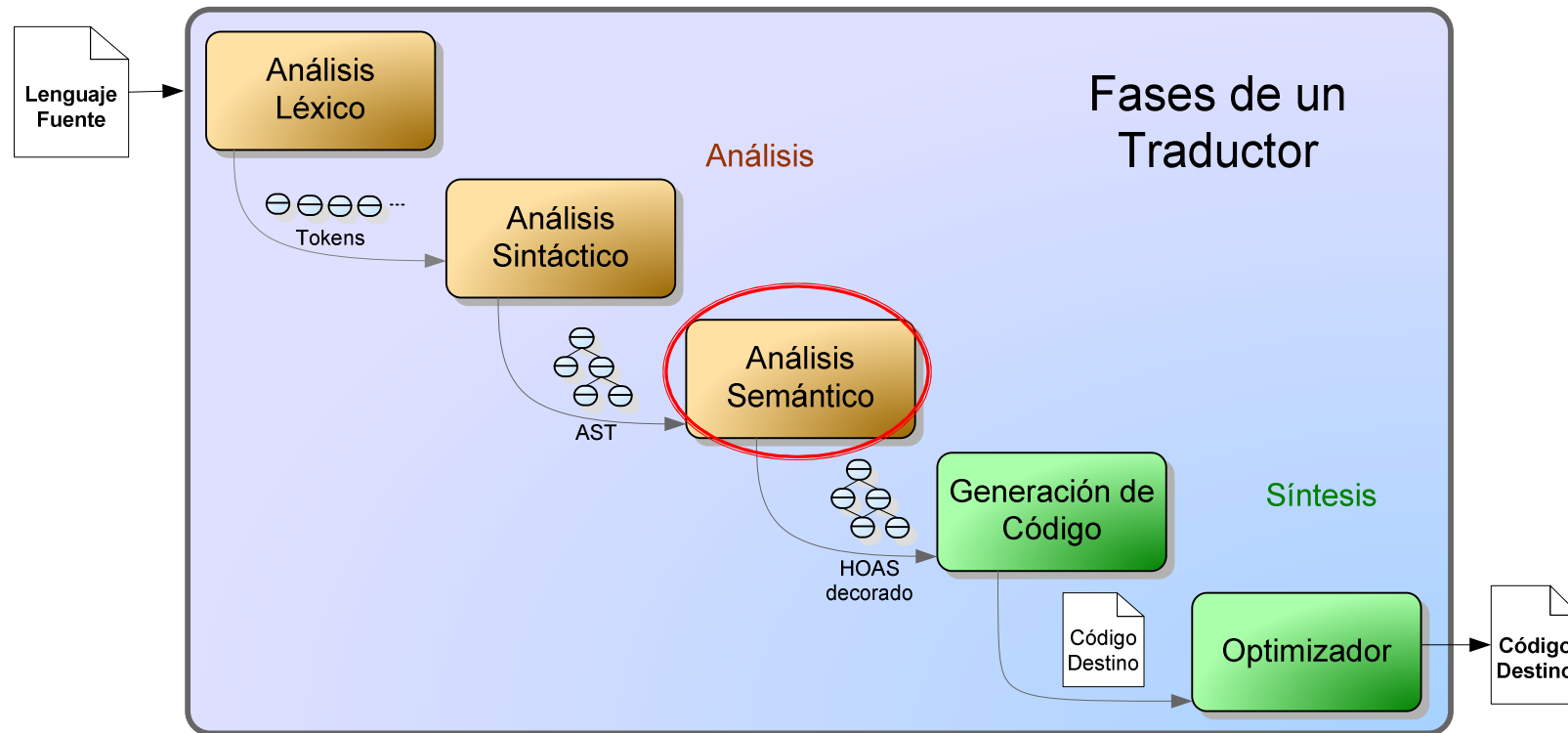
print 3 + 4 * 5, 34 - 2 + 1, 6 / 2 + 8;

Símbolo	A(x)	Dominio	Tipo

Reglas de <i>Gramática Abstracta</i>	Predicados B(p)	Reglas semánticas R(p)
print → expr*		
literalEntero:expr → string		
variable:expr → string		
aritmetica:expr → left:expr operator:string right:expr		

Fase de Identificación

Usted está aquí...



Fase de Identificación

¿Qué hace?

Errores a detectar

Errores que no han detectado el análisis léxico y sintáctico

Fase de Identificación

- 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 salida de una función debe retornar un valor
 - No puede haber un `break` fuera de un `switch`, o bucle *while* o *do-while*
- ...

Fase de Identificación

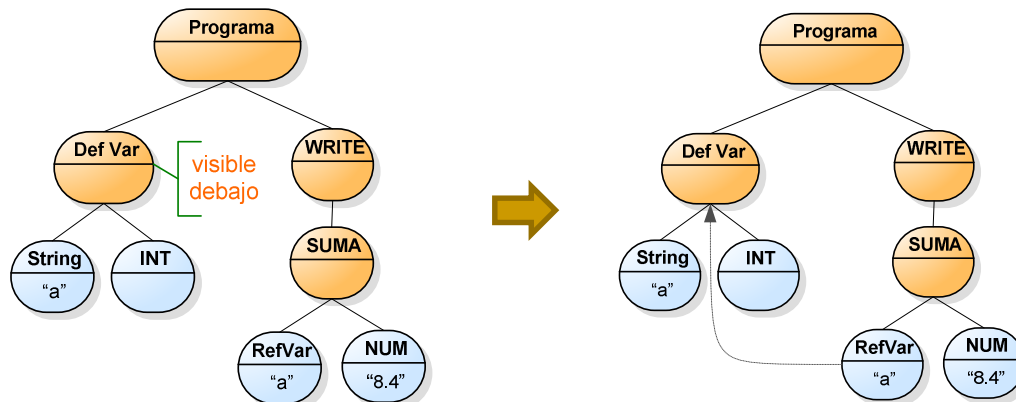
Tarea principal

- Chequeos de enlace
 - Uso de símbolos no definidos
- Chequeos de unicidad
 - Definiciones repetidas
 - Campos en una estructura
 - Enumerados
 - Sobrecarga de Funciones

Fase de
Identificación

Tarea adicional

- Enlazar las referencias a los símbolos (sus usos) con el lugar donde han sido definidos



Reglas de Ámbito (I)

¿Qué símbolos hay habitualmente en un lenguaje de programación?

Cuando se usa un símbolo ¿cómo se determina si ha sido definido?

- Reglas de Ámbito
 - Establecen el ámbito de todo símbolo en función de dónde y/o cómo se ha definido
 - Establecen las normas para los casos de solapamiento

```
// C/C++
```

```
int a;  
void f() {  
    a = 8;  
}
```

```
// C/C++
```

```
int a;  
void f() {  
    int a;  
    a = 8;  
}
```

```
// C/C++
```

```
int a;  
void f() {  
    a = 8;  
    int a;  
}
```

- ¿El ámbito de un símbolo depende sólo del lugar en donde se defina?

Errores en la Fase de Identificación (I)

```
struct Punto { x:int; y:int; };  
var b:int;  
  
main() {  
    d = b;  
    f(8);  
}  
  
var p:int;  
f(p:int) {  
    var j, w, p:int;  
}  
  
struct Punto { x:float; y:float; };  
struct Persona {  
    a: [20]Empresa;  
};  
  
f():float { }  
var b:char;
```

```
struct Punto {  
    x:int;  
    y:int;  
};  
  
var b:int;  
  
void f(p:int) {  
}  
  
main() {  
    var c:int;  
    var p:Punto;  
  
    b = f(8.5);  
    c.x = 10;  
    p.edad = 3;  
    p[5] = b;  
}
```

Errores en la Fase de Identificación (II)

En resumen

- La Fase de Identificación *solo genera* dos de los mensajes de error que puede generar un compilador
 - Símbolo no definido
 - Símbolo ya definido

Fase de Identificación

¿Cómo lo hace?

Fase de Identificación. Implementación

Implementación de la Fase de Identificación

- Depende [*mucho*] de las Reglas de Ámbito del lenguaje

Casos comunes

- Caso 1. Variables globales con ámbito parcial
- Caso 2. Variables globales con ámbito general
- Caso 3. Variables globales y locales con ámbito parcial

Caso 1. Variables Globales con ámbito parcial (I)

Reglas semánticas

- Sólo hay variables globales y su ámbito comprende las funciones posteriores
- Las funciones anteriores a la definición no pueden acceder

```
void f() {  
    a = 8;  
}
```

```
int a;
```

```
void g() {  
    a = 8;  
}
```

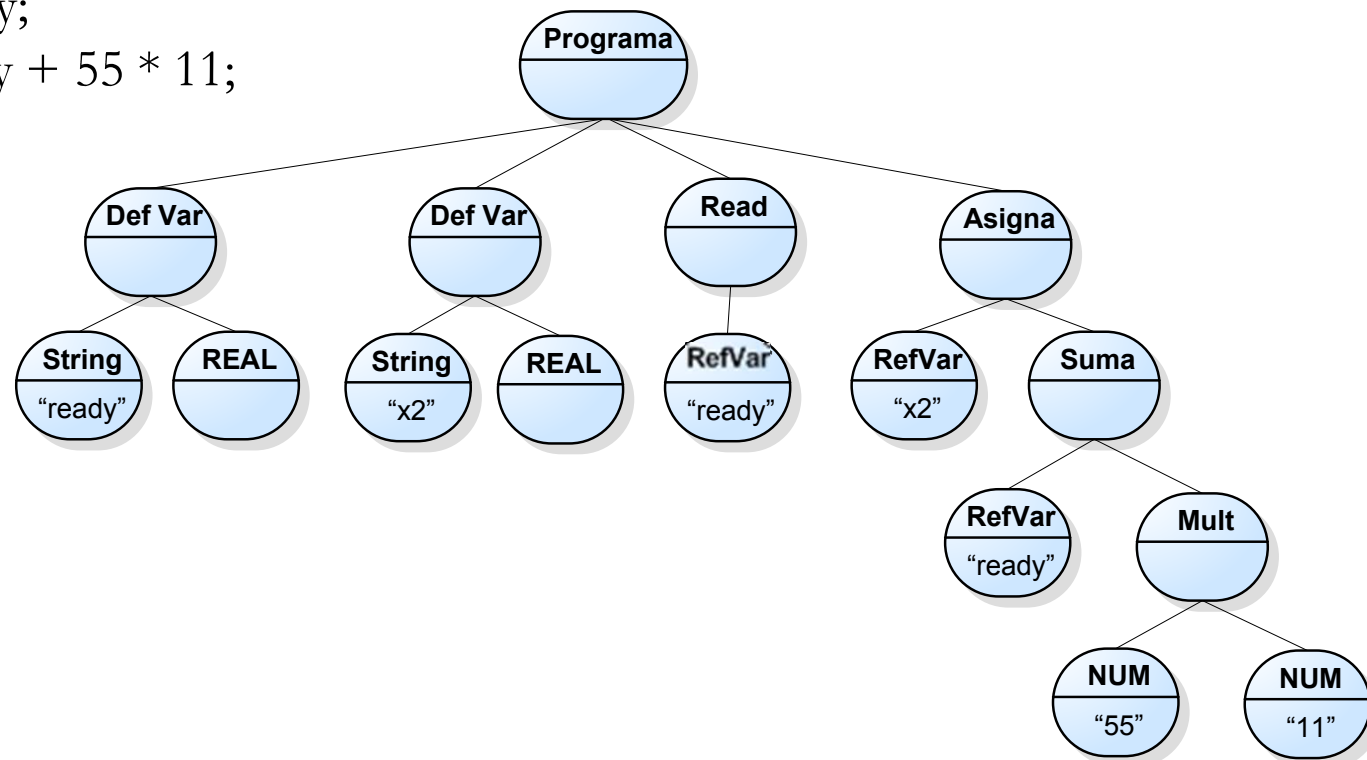
Caso 1. Variables Globales con ámbito parcial (II)

■ Implementación 1. Búsqueda Anidada

ready, x2:real;

Read ready;

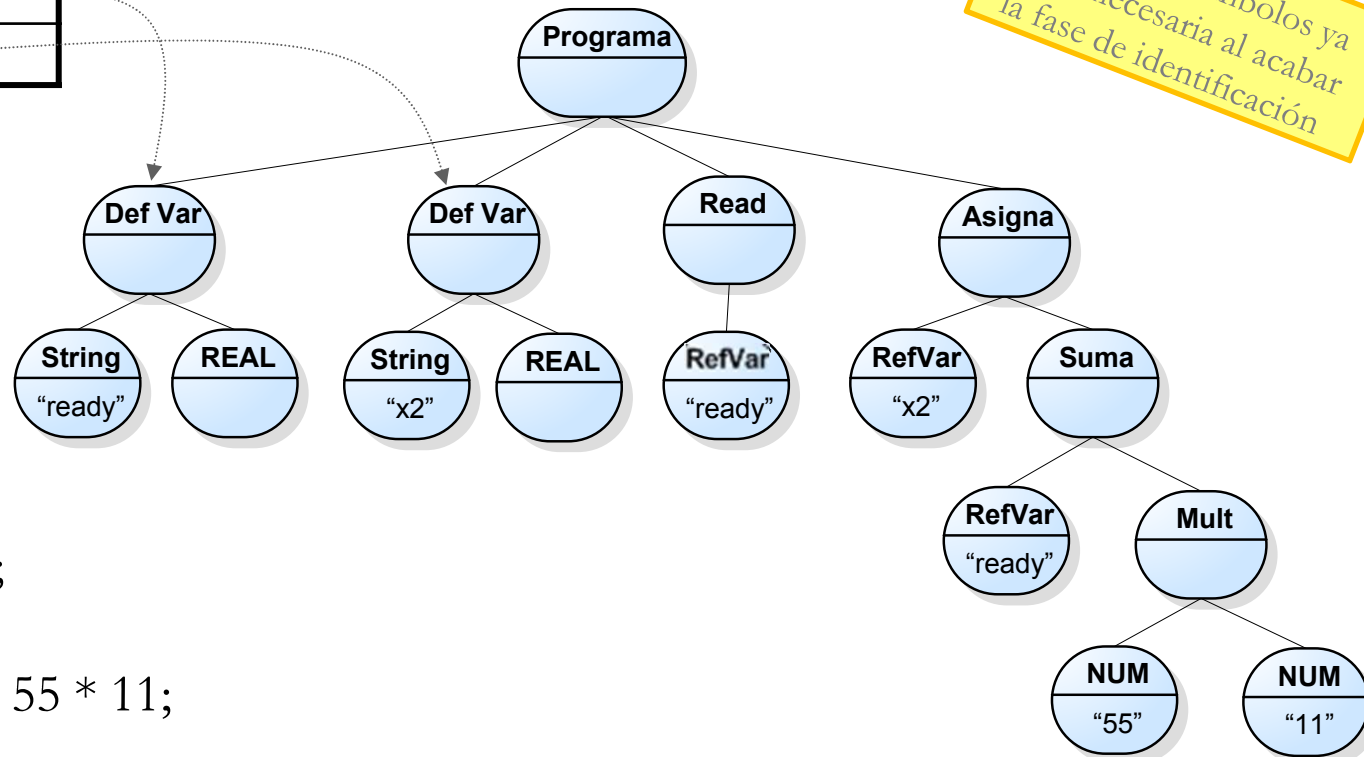
x2 = ready + 55 * 11;



Caso 1. Variables Globales con ámbito parcial (III)

■ Implementación 2. Tabla de Símbolos

Símbolo	Nodo
ready	
x2	



ready, x2:real;
Read ready;
x2 = ready + 55 * 11;

Caso 1 (III)

El sintáctico deja pasar programas inválidos como:

```
DATA int a;
```

```
CODE print b; // b no definida
```

Ejemplo

- Especificar la Fase de Identificación mediante una GAt

Símbolo	Predicados	Reglas Semánticas
programa → defvariable* sentencia*		
defvariable → nombre:string tipo		
inttype:tipo → λ		
realttype:tipo → λ		
print:sentencia → expresion		
read:sentencia → variable		
suma:expresion → left:expresion right:expresion		
variable:expresion → nombre:string		
literalint:expresion → valor:string		

Nodo/Categoría	Atributo	Dominio	Tipo

Conjuntos Auxiliares

variables Map<String, DefVariable>

Ejercicio E1

Símbolo	Predicados	Reglas Semánticas
programa → defvariable* sentencia*		
defvariable → nombre:string tipo	$\text{variables}[\text{nombre}] == \emptyset$	$\text{variables}[\text{nombre}] = \text{DefVariable}$
...		
variable:expresion → nombre:string	$\text{variables}[\text{nombre}] \neq \emptyset$	$\text{variable.definicion} = \text{variables}[\text{nombre}]$
literalint:expresion → valor:string		

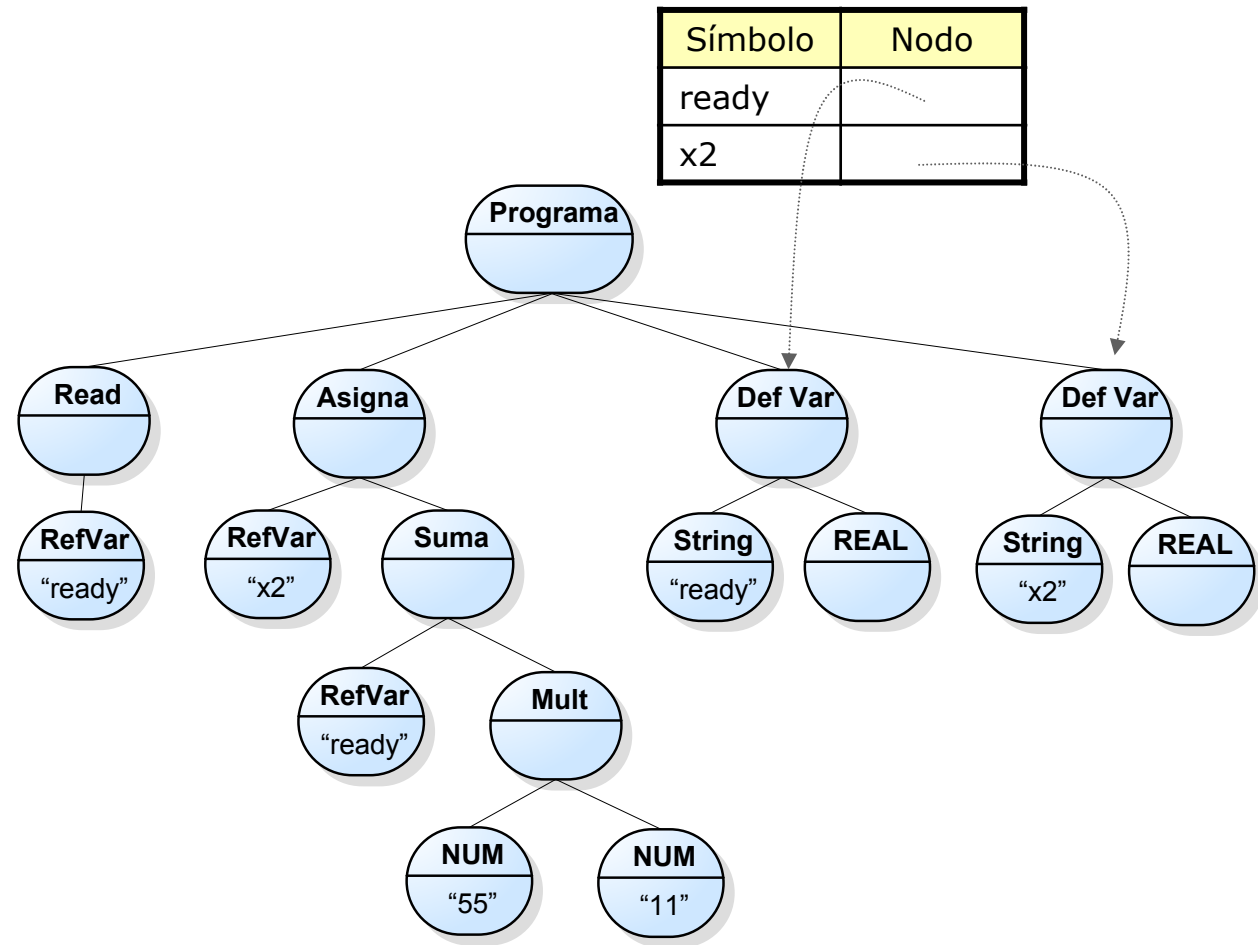
```
public class Identificacion extends DefaultVisitor {  
  
    public Object visit(DefVariable node, Object param) {  
  
        return null;  
    }  
  
    public Object visit(Variable node, Object param) {  
  
        return null;  
    }  
}
```

Caso 2. Variables Globales con ámbito general

Reglas semánticas

- Sólo variables globales cuyo ámbito es todo el fichero

Read ready;
x2 = ready + 55 * 11;
ready, x2:real;



Caso 3. Variables Globales y Locales (I)

Reglas semánticas

- Variables globales con ámbito solo en las funciones posteriores.
- Variables locales cuyo ámbito solo son las sentencias definidas posteriormente en el mismo bloque.

```
a:int;
void f() {
    local:int;

    print a;
    print local;
}
void g() {
    print local;
}
```

- Operaciones adicionales set y reset

set →

Símbolo	Nodo
a	
f	
local	

Caso 3. Variables Globales y Locales (II)

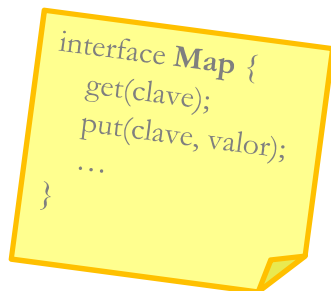
Implementación de la Tabla de Símbolos

- Sin set/reset
- Con set/reset

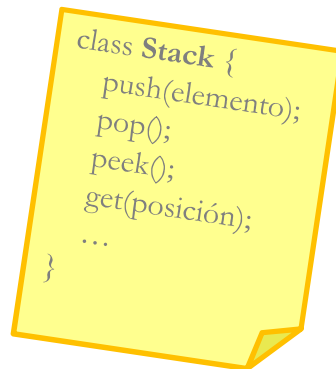
Ejercicio E2

- Implementar una Tabla de Símbolos con set/reset

```
public class TablaDeSímbolos {  
    public TablaDeSímbolos () {  
  
    }  
  
    public void inserta(String nombre, DefVar def) {  
  
    }  
  
    public DefVar busca(String nombre) {  
  
    }  
  
    public void set() {  
  
    }  
  
    public void reset() {  
  
    }  
}
```



```
interface Map {  
    get(clave);  
    put(clave, valor);  
    ...  
}
```



```
class Stack {  
    push(elemento);  
    pop();  
    peek();  
    get(posición);  
    ...  
}
```


Ejemplo E3 (I)

Sea el lenguaje

programa \rightarrow definicion*

int:tipo $\rightarrow \lambda$

real:tipo $\rightarrow \lambda$

defVar:definicion \rightarrow nombre:string tipo

defFunc:definicion \rightarrow nombre:string locales:defvar* sentencia*

print:sentencia \rightarrow expr

invoca:sentencia \rightarrow nombre:string

literalEntero:expr \rightarrow string

variable:expr \rightarrow string

suma:expr \rightarrow left:expr right:expr

Se pide

- Implementar en Java la Fase de Identificación de dicho lenguaje

Reglas de Ámbito

- Las variables globales solo pueden usarse debajo de su definición
- Las variables locales pueden usarse solo dentro de su misma función y debajo de su definición
- Una variable local tiene prioridad sobre una global
- Puede haber una variable que se llame igual que una función

Símbolo	Predicados	Reglas Semánticas
programa → definiciones:definicion*		
int:tipo → λ		
real:tipo → λ		
defVar:definicion → nombre:string tipo:tipo	¿? — ¿?	¿? $variables[nombre] = defVar$
defFunc:definicion → nombre:string locales:defVar* sentencia*	¿?	¿? $variables[nombre] = defFunc$ ¿? $variables.set()$ ¿? $variables.set(locales)$ ¿? $eval(sentencia)$ ¿? $variables.reset()$
print:sentencia → expr		
invoca:sentencia → nombre:string	¿? $variables[nombre] != null$	¿? $variables[nombre] != null$
literalEntero:expr → string		
variable:expr → nombre:string	¿? — ¿?	¿? — ¿?
suma:expr → left:expr right:expr		

Las GAt no permiten establecer orden. Usamos notación en Java

Nodo	Atributo	Dominio	Tipo Att
	¿?		

Conjuntos auxiliares

¿?

Resumen

Fase de Identificación

- Qué entra
 - El AST del sintáctico
- Qué hace
 - Encontrar 2 errores
 - Símbolo no definido
 - Símbolo ya definido
- Qué sale
 - El AST con nueva información
 - Los nodos que hacen referencia a algún símbolo tienen un nuevo atributo que enlaza con el nodo en el que se encuentra la definición de dicho símbolo

Ejercicio E4

Símbolo	Predicados	Reglas Semánticas
programa \rightarrow definiciones:definicion*		
int:tipo $\rightarrow \lambda$		
real:tipo $\rightarrow \lambda$		
defVar:definicion \rightarrow nombre:string tipo:tipo	variables.buscaActual(nombre) == null	variables[nombre] = defVar
defFunc:definicion \rightarrow nombre:string locales:defVar* sentencia*	funciones[nombre] == null	funciones[nombre] = defFunc { variables.set() visit(locales _i) visit(sentencia _i) variables.reset() }
print:sentencia \rightarrow expr		
invoca:sentencia \rightarrow nombre:string	funciones[nombre] != null	invoca.definicion = funciones[nombre]
literalEntero:expr \rightarrow string		
variable:expr \rightarrow nombre:string	variables.busca(nombre) != null	variable.definicion = variables.busca(nombre)
suma:expr \rightarrow left:expr right:expr		

Nodo	Atributo	Dominio	Tipo Att
invoca	definicion	defFunc	
variable	definicion	defVar	

Conjuntos auxiliares

Map<String, DefFunc> *funciones*

TablaSímbolos<String, DefVar> *variables*

Soluciones

Solución E1

Símbolo	Predicados	Reglas Semánticas
programa \rightarrow defvariable* sentencia*		
defvariable \rightarrow nombre:string tipo	$\text{variables}[\text{nombre}] == \emptyset$	$\text{variables}[\text{nombre}] = \text{defVariable}$
...		
variable:expresion \rightarrow nombre:string	$\text{variables}[\text{nombre}] \neq \emptyset$	$\text{variable.definicion} = \text{variables}[\text{nombre}]$
literalint:expresion \rightarrow valor:string		

```
public class Identificacion extends DefaultVisitor {
```

```
    private Map<String, DefVariable> variables = new HashMap<String, DefVariable>();
```

```
    public Object visit(DefVariable node, Object param) {
        super.visit(node, param);
```

```
        predicado(variables.get(node.getNombre()) == null, "Error. Variable ya definida");
        variables.put(node.getNombre(), node);
        return null;
    }
```

```
    public Object visit(Variable node, Object param) {
        DefVariable definicion = variables.get(node.getNombre());
        predicado(definicion != null, "Error. Variable no definida");
        node.setDefinicion(definicion); // Enlazar referencia con definición
```

```
        return null;
```

```
    }
}
```

Conjuntos Auxiliares

variables Map<String, DefVariable>

Nodo/Categoría	Atributo	Dominio	Tipo
variable	definicion	DefVariable	Sintetizado

Solución E2

```
public class TablaDeSímbolos {  
  
    public TablaDeSímbolos() {  
        set();  
    }  
  
    public void inserta(String nombre, DefVar def) {  
        pilaAmbitos.peek().put(nombre, def);  
    }  
  
    public DefVar busca(String nombre) {  
        for (int i = (pilaAmbitos.size() - 1); i >= 0; i--) {  
            Map<String, DefVar> ambito = pilaAmbitos.get(i);  
            DefVar def = ambito.get(nombre);  
            if (def != null)  
                return def;  
        }  
        return null;  
    }  
  
    public void set() {  
        pilaAmbitos.push(new HashMap<String, DefVar>());  
    }  
  
    public void reset() {  
        pilaAmbitos.pop();  
    }  
  
    private Stack<Map<String, DefVar>> pilaAmbitos = new Stack<Map<String, DefVar>>();  
}
```

Lo suyo sería
haberlo hecho
con genericidad...

¿Falta algo aquí?