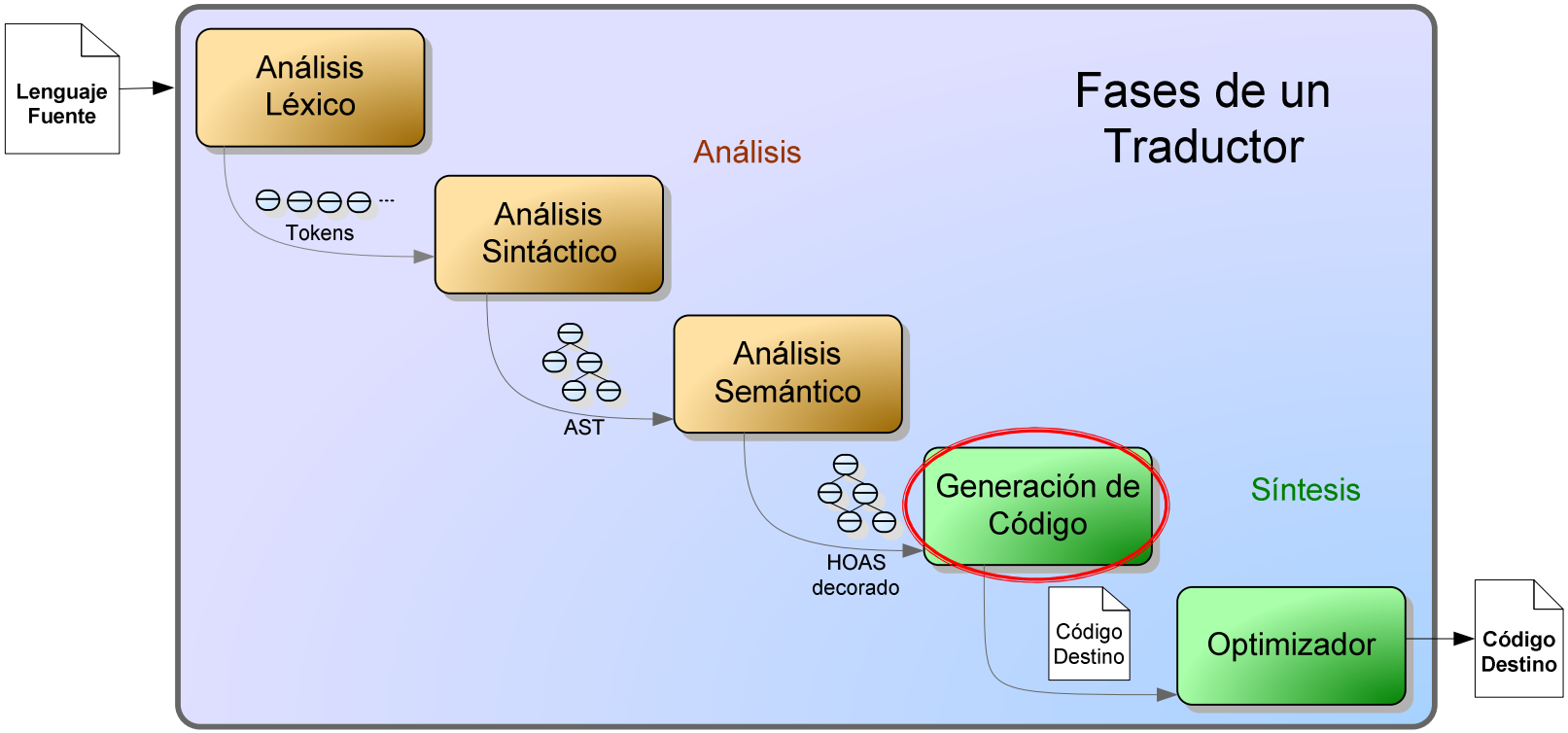
Generación de Código (III)

Diseño de Lenguajes de Programación
Ingeniería Informática
Universidad de Oviedo
(v1.4)

Raúl Izquierdo Castanedo

Usted está aquí…



Repaso

La fase de Generación de Código se encarga de determinar...

- Cómo se representan los Datos
  - Representar los tipos abstractos de alto nivel con las limitaciones de la plataforma de destino
- Cómo se traduce el Código
  - Traducir las sentencias de alto nivel a secuencias de instrucciones de la plataforma de destino

Subfases:

- Gestión de Memoria 
- Selección de Instrucciones 

Selección de Instrucciones
Repaso [muy] rápido

Selección de Instrucciones. ¿*Cómo* se hace?

Generación Inductiva

- El código generado para un programa se obtiene mediante la unión del código generado por sus estructuras

Metalenguaje: Especificación de Código

- Conjunto de *funciones de código* que especifican la traducción al lenguaje destino de todo programa válido

$$f_p : P \rightarrow Instrucción*$$

Elementos de una Función de Código

1) Nombre

direccion[[expr]]
valor[[expr]]

2) Conjunto de Plantillas de Código
   - Una por cada nodo de la categoría
     - Especifica el código al que hay que traducir la estructura (nodo) en términos del código obtenido en la traducción de sus subestructuras (hijos)

Notación de las Plantillas

Se seguirá la siguiente notación

- La plantilla genera su contenido *literalmente*

```
CALL func.nombre        // Genera "CALL func.nombre" (no se evalúa)
```

- Evaluación de expresiones

```
CALL { func.nombre }    // Genera "CALL f" (suponiendo que f es el nombre)
```

- Llamadas a funciones de código

```
función[[nodo]]         // Genera las instrucciones de dicha función
```

- Pseudocódigo *si/sino*

```
valor [[ exprLogica → left:expresion operador:string right:expresion ]] =
                valor[[left]]
                valor[[right]]
                Si operador == "&&"
                    AND
                Si operador == "||"
                    OR
```

- Instrucciones *genéricas*
  - Abreviatura para la siguiente estructura habitual

*instrucción*_{<expresión_de_tipo>}

```
Si expr_de_tipo == tipo_entero
    instrucciónI
sino si expr_de_tipo == tipo_real
    instrucciónF
sino si expr_de_tipo == tipo_byte
    instrucciónB
```

Anteriormente…

```
var i:int;

f() {
   print 1;
}

fecha():int {
   return 1492;
}

main() {

   if (37) {
      print 750;
   }

   print fecha();
   i = fecha();
   fecha();
}
```

```
call main
halt

f:
' print 1;
push 1
out

ret 0, 0, 0


fecha:
' return 1492;
push 1492
ret 2, 0, 0


main:

' if(37) {
push 37
jz else1

' print 750
push 750
out

jmp finelse1   ' opcional en este caso

else1:

finelse1:      ' opcional en este caso

' print fecha();
call fecha
out

' i = fecha();
pusha 0
call fecha
store

' fecha();
call fecha
pop


ret 0, 0, 0
```

Ejercicio E1

programa → definicion*

enum Ambito { GLOBAL, LOCAL, PARAMETRO};

definicionVariable:definicion → *nombre*:string tipo ambito

definicionFuncion:definicion → *nombre*:string *parametros*:definicionVariable* *retorno*:tipo *locales*:definicionVariable* *sentencias*:sentencia*

tipoInt:tipo → λ

tipoReal:tipo → λ

tipoChar:tipo → λ

tipoVoid:tipo → λ

return:sentencia → *expr*:expresion

invocaProc:sentencia → *nombre*:string *args*:expresion*

variable:expresion → *nombre*:string

constanteReal:expresion → *valor*:string

invocaFunc:expresion → *nombre*:string *args*:expresion*

Soluciones

Solución E1 (I)

Función de Código	Plantillas de Código
run : **programa** → Instruccion*	run [[**programa** → *definiciones*:definicion*]] = CALL main HALT define[[definiciones$_i$]]
define : **definicion** → Instruccion*	define[[**definicionVariable** → *nombre*:String tipo ambito]] = define[[**definicionFuncion** → *nombre*:string *parametros*:definicionVariable* *retorno*:tipo *locales*:definicionVariable* *sentencias*:sentencia*]] = {nombre}: ENTER {∑locales$_i$.tipo.size} ejecuta[[sentencias$_i$]] Si retorno == VOID RET 0, {∑locales$_i$.tipo.size}, {∑parametros$_i$.tipo.size}
ejecuta : **sentencia** → Instruccion*	ejecuta [[**return** → *expr*:expresion]] = si expr ≠ null valor[[expr]] RET {expr.tipo.size}, {∑return.***función***.locales$_i$.tipo.size}, {∑return.función.parametros$_i$.tipo.size} sino RET 0, {∑return.función.locales$_i$.tipo.size}, {∑return.funcion.parametros$_i$.tipo.size}

Atributo "función" añadido en la fase de Comprobación de Tipos

Solución E1 (II)

Función de Código	Plantillas de Código
ejecuta : **sentencia** → Instruccion*	ejecuta [[**invocaProc** → *nombre*:string *args*:expresion*]] = valor[[$args_i$]] CALL {nombre} si invocaProc.***definición***.retorno != VOID POP$_{\text{<invocaProc.definición.retorno>}}$
valor: **expresion** → Instruccion*	valor [[**variable** → *nombre*:string]] = direccion[[variable]] LOAD$_{\text{<variable.definicion.tipo>}}$ valor [[**constanteReal** → *valor*:string]] = PUSHF {valor} valor [[**invocaFunc** → *nombre*:string *args*:expresion*]] = valor[[$args_i$]] CALL {nombre}
direccion: **expresion** → Instruccion*	direccion [[**variable** → *nombre*:string]] = si variable.definicion.ambito == GLOBAL PUSHA {variable.definición.dirección} sino PUSH BP PUSH {variable.definición.dirección} ADD

Atributo "definición" añadido en la Fase de Identificación