

NO:

DATE:

1-Mayo-2025 Juan Antonio Pérez Sánchez

Traductores de Lenguajes II - Julio Esteban Valdez López

Resumen 3.- Capítulo 6.1 al 6.7

Los nodos en un árbol sintáctico, representan construcciones en el programa fuente; las hojas de un nodo representan las componentes significativas de una construcción.

Un grafo acíclico Dirigido - GDA. Para una expresión identifica las a las subexpresiones comunes

6.1.1 Un grafo que d árbol sintáctico para una expresión, un GDA tiene hojas que corresponden a los operandos atómicos y códigos interiores que corresponden a los operadores. La diferencia es que un nodo  $N$  en un GDA tiene más de un padre si  $N$  representa a una subexpresión común; en un árbol sintáctico, el arbol para la subexpresión común, se replica tantas veces como aparezca la subexpresión original. Por ende un GDA no solo representa las expresiones de la manera más breve, sino que también proporciona pistas importantes al compilador.

6.1.2 A menudo los nodos de un arbol sintáctico o GDA se almacenan en un arreglo de registros.

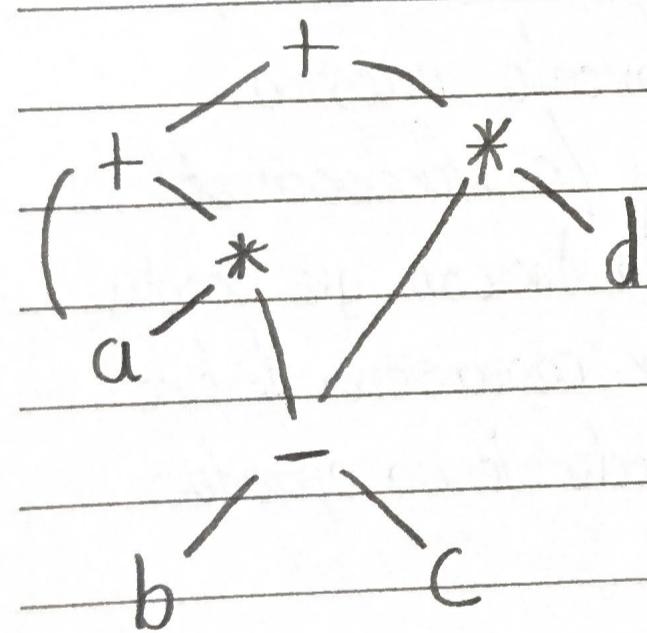
Cada fila del arreglo representa a un registro y por lo tanto a un nodo. En cada registro, el primer campo es un código de operación, el cual contiene el valor léxico y los nodos

Juan Cintiaño Poy, fuerza

Traductores de Lenguajes II - Julio Esteban Valdez López

Interares tienen 2 campos interiores adicionales que indican los hijos izquierdo y derecho.

c) Código de 3 direcciones: Hay máximo un operador de lado derecho de una instrucción, no se permiten expresiones aritméticas acumuladas.



$$t_1 = b - c$$

$$t_2 = a * t_1$$

$$t_3 = a + t_2$$

$$t_4 = t_1 * d$$

$$t_5 = t_3 + t_4$$

b) Código de 3 direcciones

a) DAG

El código de 3 direcciones se basa en 2 conceptos: Direcciones e instrucciones. En términos de orientación de objetos, estos conceptos corresponden a la clase y los diversos tipos de direcciones e instrucciones corresponden a subclases apropiadas.

Una dirección puede ser una de las siguientes opciones:

- Un nombre
- Una constante
- Un valor temporal

generado por un compilador

NO:

DATE: 1-Mayo-2025 Juan Antonio Peña Jiménez

Traductores de Lenguajes II - Dr. Esteban Valdez

### 6.3 Tipos y declaraciones.

Las aplicaciones de los tipos pueden agruparse mediante la comprobación y la traducción:

- La comprobación de tipos utiliza reglas lógicas para razonar acerca del comportamiento de un programa en tiempo de ejecución.
- Aplicaciones de traducción. A partir del tipo de un nombre, un compilador puede determinar el almacenamiento necesario para ese nombre en tiempo de ejecución. La información del tipo también se necesita para calcular la dirección que denota la referencia a un arreglo, para insertar conversiones de tipo explícitas, y para elegir la versión correcta de un operador aritmético, entre otras cosas.

Expresiones de tipo: Los tipos tienen estructura, a la cual representamos mediante el uso de las expresiones de tipos:

Una expresión de tipos es un tipo básico o se forma mediante la aplicación de un operador llamado constructor de tipos a una expresión de tipos. Los conjuntos de tipo básicos y los constructores dependen del lenguaje que se va a comprobar

arreglo

2

arreglo

3

integer

Expresión de tipos para  
int [2][3]

Juan Antonio Pérez Juárez

Traductores de Lenguajes II - Julio Esteban Valle López

tipo = arreglo [2, arreglo [3, integer]]

tamaño = 24

N : tipo = integer  
tamaño = 4

t = integer  
w = 4

int

C. tipo = arreglo [2, arreglo [3, integer]]  
tamaño = 24

[2]

C. tipo = arreglo [3, integer]  
tamaño = 12

E

Traducción orientada

por la sintaxis de los tipos  
de arreglos.

[3]

tipo = integer  
tamaño = 4

6.5 Comprobación de tipos: Para la comprobación de tipos, un compilador debe asignar una expresión de tipos a cada componente del programa fuente. Despues el compilador debe determinar que estas expresiones de tipos se conforman a una corrección de reglas lógicas, conocido como el sistema de tipos para el lenguaje fuente.

La comprobación de tipos tiene el potencial de atrapar errores en los programas. En principio, cualquier comprobación puede realizarse en forma dinámica, si el código de destino lleva el tipo de un elemento, junto con el valor del elemento.

6.5.1 Sobre carga de funciones y operadores: La sobre carga de un símbolo tienen diferentes significados, dependiendo de su contexto.

NO:

DATE: 05-Mayo-2025 Juan Antonio Pérez Jiménez

Técnicas de Lenguajes II - Julio Esteban Voldz

## 6.6 Flujo de Control.

La traducción de instrucciones como `if-else` y `while` está enlazada a la traducción de expresiones booleanas.

En los lenguajes de programación, las expresiones booleanas se utilizan con frecuencia para:

1. - Alterar el flujo de control
2. - Calcular Valores lógicos

Las expresiones booleanas están compuestas de los operadores booleanos que denotamos como:  $\&$  → AND

$\|$  → OR

$!$  → NOT

### 6.6.2 Código de Corto Circuito.

El código de corto circuito (o de salto), los operadores booleanos se traducen en saltos.

Los mismos operadores no aparecen en el código; en vez de ello, el valor de una expresión booleana se representa mediante una posición en la secuencia de código.

5-Mayo-25

Juan Antonio Pérez Sánchez

Tutoriales de Lenguajes I - Alba Esteban Valdiz López

$$S \rightarrow id = E; \quad S. \text{codigo} = E. \text{codigo} \parallel$$

$$E \rightarrow E_1 + E_2 \quad \text{gen}(\text{tpe}, \text{get}(id), \text{lexema})' = 'E. \text{dir}'$$

$$| - E_1$$

$$| (E_1)$$

$$| id$$

$$E. \text{dir} = E_1. \text{codigo} \parallel E_2. \text{codigo} \parallel$$

$$\text{gen}(\text{tpe}, \text{get}(id), \text{lexema})' = 'E. \text{dir}'$$

$$E. \text{dir} = E_1. \text{codigo} \parallel E_2. \text{codigo} \parallel$$

$$\text{gen}(E. \text{dir}') = E_1. \text{dir}' \parallel E_2. \text{dir}'$$

Tiene ambigüedad

y recursión por

izquierda y

derecha

$$E. \text{dir} = \text{new Temp}()$$

$$E. \text{codigo} = E_1. \text{codigo} \parallel \text{gen}(E. \text{dir}') = 'menos' E. \text{dir}$$

$$E. \text{dir} = E_1. \text{dir}$$

$$E. \text{codigo} = E_1. \text{codigo}$$

$$E. \text{dir} = \text{tpe}. \text{get}(id). \text{lexema}$$

$$E. \text{codigo} = "$$

○

○

○

○

—

$$P \rightarrow S$$

$$S. \text{siguiente} = \text{nuevaetiqueta}()$$

$$S \rightarrow \text{assign}$$

$$P. \text{codigo} = S. \text{codigo} \parallel \text{etiqueta}(S. \text{siguiente})$$

$$S \rightarrow ; f(B) S_1$$

$$S. \text{codigo} = \text{assign. codigo}$$

$$B. \text{true} = \text{nuevaetiqueta}()$$

$$B. \text{false} = S_1. \text{siguiente} = S. \text{siguiente}$$

$$S \rightarrow if(B) S_1 \text{ else } S_2$$

$$S. \text{codigo} = B. \text{codigo} \parallel \text{etiqueta}(B. \text{true}) \parallel S_1. \text{codigo}$$

$$B. \text{true} = \text{nuevaetiqueta}()$$

$$B. \text{false} = \text{nuevaetiqueta}()$$

NO:

DATE: 5-Mayo-25

Juan Lintonio Pérez Jerez

S<sub>1</sub>.siguiente = S<sub>2</sub>.siguiente; S<sub>1</sub>.siguiente

S<sub>2</sub>.codigo = B.codigo || etiqueta(B.true) || S<sub>2</sub>.codigo  
|| gen('goto' S<sub>1</sub>.siguiente)  
|| etiqueta(B.false) || S<sub>2</sub>.codigo

S → while (B) si

inicio = nuevaetiqueta()

B.true = nuevaetiqueta()

B.false = S<sub>1</sub>.siguiente

S<sub>1</sub>.siguiente = inicio

S<sub>2</sub>.codigo = etiqueta(inicio) || B.codigo

|| etiqueta(B.true) || S<sub>1</sub>.codigo

|| gen('goto' inicio)

S → S<sub>1</sub> S<sub>2</sub>

S<sub>1</sub>.siguiente = nuevaetiqueta()

S<sub>2</sub>.siguiente = S<sub>1</sub>.siguiente

S.codigo = S<sub>1</sub>.codigo || etiqueta(S<sub>1</sub>.siguiente)

|| S<sub>2</sub>.codigo

### Expresiones Booleanas

B → B<sub>1</sub> || B<sub>2</sub>

B<sub>1</sub>.true = B.true

B<sub>1</sub>.false = nuevaetiqueta()

Es lo cs como

B<sub>2</sub>.true = B.true

un OR

B<sub>2</sub>.false = B.false

B.codigo = B<sub>1</sub>.codigo || etiqueta(B<sub>1</sub>.false) ||| B<sub>2</sub>.codigo

B → B<sub>1</sub> & B<sub>2</sub>

B<sub>1</sub>.true = nuevaetiqueta()

AND

B<sub>1</sub>.false = B.false

From Control Bigg Lang

NO:

DATE: 5-5-25

$B_1.\text{true} = B.\text{true}$

$B_2.\text{false} = B.\text{false}$

$B.\text{codigo} = B_1.\text{codigo} \parallel \text{ct.(getul}(B_1.\text{true})) \parallel B_2.\text{codigo}$

$B \rightarrow !B_1$

$B.\text{true} = B.\text{false}$

$B.\text{false} = B.\text{true}$

$B.\text{codigo} = B_1.\text{codigo}$

$B \rightarrow E_1 \text{ rel } E_2$

$B.\text{codigo} = E_1.\text{codigo} \parallel E_2.\text{codigo}$

$\parallel \text{gen('if' } E_1.\text{dir rel.op } E_2.\text{dir 'goto' } B.\text{true})$

$\parallel \text{gen('goto' } B.\text{false})$

$B \rightarrow \text{True}$

$B.\text{codigo} = \text{gen('goto' } B.\text{true})$

$B \rightarrow \text{False}$

$B.\text{codigo} = \text{gen('goto' } B.\text{false})$