Tema 9

Análisis Semántico

Universidad Europea

LAUREATE INTERNATIONAL UNIVERSITIES

Comprobación de Tipos

Prof. Leopoldo Santos Santos 2020

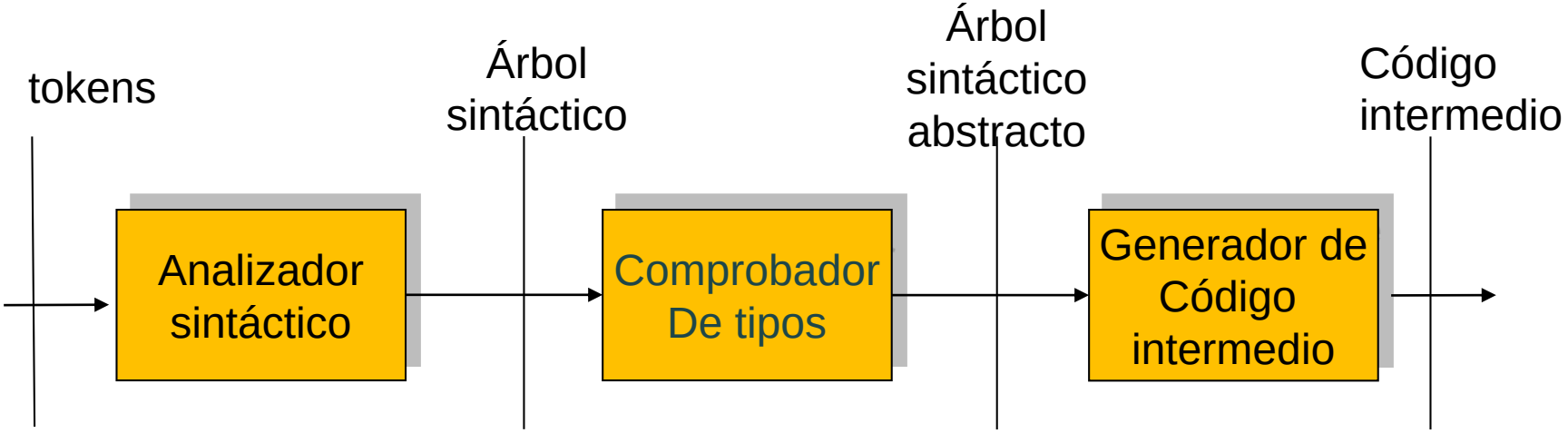
Índice

- Introducción
- Sistemas de tipos
  - Expresiones de tipos
  - Sistemas de tipos
  - Comprobación estática y dinámica de tipos
- Especificación de un comprobador de tipos sencillo
- Operadores sobrecargados
- Equivalencia de expresiones de tipo

Comprobación de Tipos

- Un comprobador de tipos se asegura de que el tipo de una construcción coincida con el previsto en su contexto

- Ubicación:



Expresiones de tipos I

- Representan el tipo de las construcciones
  - Especificaciones del lenguaje para operaciones:
    - Tipo de Expresión resultante de aplicar operadores aritméticos
    - Resultado de aplicar operador &
    - Tipo de llamada a función, etc.
- Pueden ser
  - Un tipo básico
    - Boolean, Char, Integer, Real, Error_tipo, Vacio…
  - Un constructor de tipos aplicado a expresiones de tipos
    - Tipo array de variables de tipo básico
    - Tipo puntero a variable
    - Tipo registro, etc

Expresiones de tipos II

- Expresiones de tipos
  1. Tipos básicos
  2. El nombre de un tipo es una expresión de tipo
  3. Constructores de tipos y expresiones de tipos son tipo

  a) Arrays

     Si T es una expresión de tipo entonces array (I, T) es una expresión de tipo que indica una matriz con elementos de tipo T y conjunto de índices I

     Var A: array[1..10] of Integer es array (1..10, Integer)

  b) Productos

     Si T1 y T2 son expresiones de tipo, su producto cartesiano T1 x T2 es una expresión de tipo (x es asociativo por la izquierda)

Expresiones de tipos III

c) Registros

Tiene como tipo el producto de los tipos de los campos. El constructor de tipo “record” se aplica a una tupla formada con nombres de campos y tipos de campos. Ej:

```
type fila = RECORD
            direccion: integer;
            lexema: array [1..15] of Char;
        END;

var tabla: array [1..10] of fila
```

fila tiene el tipo:

record ((dirección x integer) x (lexema x array(1..15, Char)))

Tabla tiene el tipo:

array(1..10, record ((dirección x integer) x (lexema x array(1..15,char)))

o: array(1..10, *Tipo_Fila*)

Expresiones de tipos IV

d) Punteros

Si T es una expresión de tipo, Pointer (T) es una expresión de tipo que es un puntero a un objeto de tipo T. Ej.:

```
Var p: ^fila;
```

p es de tipo: pointer (fila)

e) Funciones

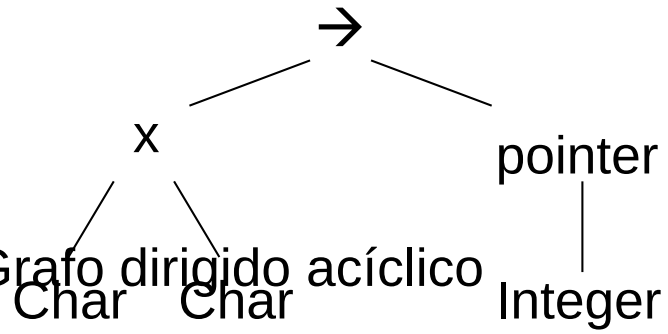
Si T_1 y T_2 son expresiones de tipos, entonces $T_1 \rightarrow T_2$ es la expresión de tipos de una función que toma argumentos de T_1 y los transforma en T_2

Function f (a, b : Char) : ^Integer;

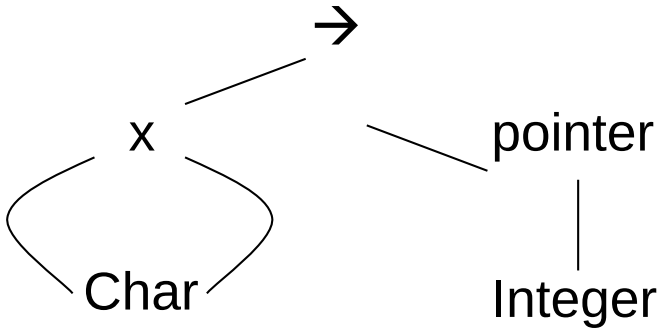
f tiene el tipo: (Char x Char) → pointer (Integer)

Expresiones de tipos V

- Representaciones de expresiones de tipos
  - Representación en árbol



  - Representación en Grafo dirigido acíclico



Sistemas de tipos

- Sistema de tipos: conjunto de reglas para asignar expresiones de tipos a construcciones de un programa
  - Un comprobador de tipos implanta un sistema de tipos
  - Se pueden implementar en la definición dirigida por la sintaxis

```
VAR x: RECORD
        p_real: real;
        p_imag: real;
END;
VAR y: RECORD
        p_imag: real;
        p_real: real;
END;
x=y;                  INCORRECTO
x.p_real=y.p_imag;    CORRECTO
```

- Compilación
  - Estática: Aquella que es realizada por el compilador antes de la ejecución del programa
  - Dinámica: Aquella que es realizada al ejecutar el programa objeto

Comprobación estática y dinámica II

- Sistema de tipos seguro (*sound type*): no necesita comprobar dinámicamente errores de tipos

- Lenguajes fuertemente tipificados: garantizan que los programas que aceptan se ejecutarán sin errores de tipo

- Algunas comprobaciones sólo pueden ser dinámicas:

```
Tabla: array[0..255] of Char;
i: Integer;
<< se asigna valor a i >>
Tabla [i] := 3;
```

  – El compilador no puede garantizar estáticamente que el valor de la variable i no exceda los límite del array tabla

Un Comprobador de Tipos Sencillo

- Un lenguaje sencillo
- Comprobación de tipos en expresiones
- Comprobación de tipos en proposiciones
- Comprobación de tipos de funciones

Un lenguaje sencillo I

- Ejemplo de lenguaje, con todos los identificadores declarados antes de uso
  - Objetivo: comprobar el tipo de toda expresión
- Tipos del lenguaje
  - Tipos básicos
    - Char
    - Integer
    - Error_Tipo
  - Tipos complejos
    - Array[n] of T es de tipo array (T, 1..n)
    - ^T es de tipo pointer (T)

Un lenguaje sencillo II

- Gramática del lenguaje

P → D ; E
D → D ; D | id : T
T → char | integer | array [num] of T | ^T
E → literal | num | id | E mod E | E [E] | E^

– [illegible]
– T es el tipo
– E es la expresión

Un lenguaje sencillo III

- Acciones semánticas de construcción de tipos

P → D ; E	
D → D ; D	
D → id : T	{ añadeTipo (id.entrada, T.tipo) }
T → char	{ T.tipo := char }
T → integer	{ T.tipo := integer }
T → ^T_1	{ T_0.tipo := pointer (T_1.tipo) }
T → array [num] of T	{ T_0.tipo := array(1..num.val, T_1.tipo) }

Un lenguaje sencillo IV

- Acciones semánticas de verificación de tipos en expresiones
  - Constantes

E → literal	{ E.tipo := Char }
E → num	{ E.tipo := Integer }

  - Identificadores

E → id	{ E.tipo := buscaTipo (id.entrada) }

  - Operadores

E → E mod E	{ if (E$_1$.tipo = Integer) and (E$_2$.tipo = Integer) E$_0$.tipo = Integer else E$_0$.tipo = Error_Tipo }

Un lenguaje sencillo V

– Arrays

$E \rightarrow E\ [\ E\ ]$ { if (E_2.tipo = Integer) and
(E_1.tipo = array(s, t) →)
E_0.tipo = t
else E_0.tipo = Error_Tipo }

– Punteros

$E \rightarrow E\ \hat{}$ { if (E_1.tipo = pointer(t))
E_0.tipo = t
else E_0.tipo = Error_Tipo }

Tipos en sentencias

- Por defecto, una sentencia es de tipo nulo (vacío), salvo si es errónea
  - Proposiciones if y while

S → id := E { if (id.tipo = E.tipo) then S_0.tipo = vacío
else S_0.tipo := Error_Tipo }

S → if E then S { if (E.tipo = boolean) then S.tipo = S_1.tipo
else S_0.tipo := Error_Tipo }

S → while E do S { if (E.tipo = boolean) then S.tipo = S_1.tipo
else S_0.tipo := Error_Tipo }

S → S ; S { if (S_1.tipo = vacío) and (S_2.tipo = vacío)
then S_0.tipo = vacío
else S_0.tipo := Error_Tipo }

Tipos en sentencias

– Ampliar G para permitir declaración de funciones

T → T '→' T	{ T_0.tipo = T_1.tipo → T_2.tipo }

Ej.: int f(double x, char y) { ... }

tipo de f: double x char → int

tipos argum. tipo devolución

– Aplicación de argumentos a una función

E → E (E)	{ if (E_2.tipo = s) and (E_1.tipo = s→t) then E_0.tipo = t else E_0.tipo := Error_Tipo }

Conversiones de tipos

- Algunos operadores pueden aplicarse a operandos de distintos tipos:
  - x + y ?
  - Si el tipo de x es double y el de y int, tipo resultado? (afecta al código a generar)

```
E → E op E        { if (E1.tipo = Integer) and
                        (E2.tipo = Integer) then E0.tipo = Integer
                    else if  (E1.tipo = Integer) and
                        (E2.tipo = real) then E0.tipo = real
                    else if  (E1.tipo = real) and
                        (E2.tipo = Integer) then E0.tipo = real
                    else if  (E1.tipo = real) and
                        (E2.tipo = real) then E0.tipo = real
                     else E0.tipo = tipo_error}
```

Tipos sobrecargados

- Algunos operadores y funciones tienen distinto significado según su contexto:
  - (4 + “a”) y (4+’a’) son expresiones distintas en Java
  - function “*” (k,j: integer) return complex
  - function “*” (x,y: complex) return complex
  - No siempre es posible resolver sólo con los operandos:

    “*” Puede tener los tipos posibles:

    integer x integer → integer

    integer x integer → complex

    complex x complex → complex

    Así:

    3*5 podría ser integer o complex, según contexto

    en la expresión 2*(3*5) tiene tipo (3*5) → integer

    en la expresión z*(3*5) tiene tipo (3*5) → complex

Tipos sobrecargados

– Se generaliza el conjunto de tipos posibles de una expresión:

E' → E	{ E'.tipos=E.tipos}
E → id	{ E.tipos=consulta(id.entrada)}
E → E (E)	{E_0.tipos= {t | existe un tipo s en E_2.tipos tal que s→ t, con t en E_1.tipos}

- Se supone que la tabla de símbolos puede contener el conjunto de posibles tipos
- El tipo conjunto vacío se asimila con tipo_error

Tipos sobrecargados

- Algunos lenguajes, como Ada, obligan a que una expresión finalmente tenga un tipo único, sino es un error
- Se introducen un nuevo atributo: "**único**"

E' → E { E'.tipos:=E.tipos
E.unico:= if E'.tipos={t} then t
else tipo_error}

E → id { E.tipos:=consulta(id.entrada)}

E → E (E) {E_0.tipos:= {s' | existe un tipo s en
E_2.tipos tal que s→s' está en E_1.tipos
t=E.unico
S= {s tal que s en E_2.tipos y s→ t en E_1.tipos}
E_2.unico:=if S== {s} then {s}, else tipo_error
E_1.unico:=if S== {s} then s→ t, else tipo_error }

-

- Implementación en dos pasadas para evaluar "tipos" y "único"

Equivalencia de Expresiones de Tipos

- Equivalencia estructural
- Nombres para expresiones de tipos
- Ciclos en las representaciones de tipos

Equivalencia estructural I

- Dos expresiones de tipos son estructuralmente equivalentes si son el mismo tipo básico o se forman aplicando el mismo constructor de tipos sobre expresiones de tipos estructuralmente equivalentes
  - Integer es equivalente a Integer
  - pointer (Integer) es equivalente a pointer (Integer)
  - …

- Las expresiones estructuralmente equivalentes se representan mediante árboles o grafos dirigidos acíclicos iguales

Equivalencia estructural II

- Algoritmo para comprobar la equivalencia:

```
Function Equivale (s, t) : boolean
    if s y t son del mismo tipo básico then return true
    else if s = array(s1, s2) and t = array (t1, t2) then
            return Equivale (s1, t1) and Equivale (s2, t2)
    else if s = s1 x s2 and t = t1 x t2 then
            return Equivale (s1, t1) and Equivale (s2, t2)
    else if s = pointer (s1) and t = pointer (t1) then
            return Equivale (s1, t1)
    else if s = s1 → s2 and t = t1 → t2 then
            return Equivale (s1, t1) and Equivale (s2, t2)
    else return false
```

  – Algunas comprobaciones pueden “relajarse” (arrays…)

Nombres de expresiones de tipos I

- Algunos lenguajes permiten nombrar tipos

```
Type enlace = ^nodo;
Var siguiente : enlace;
      ultimo: enlace;
      p : ^nodo;
      q, r : ^nodo;
```

- ¿Tienen igual tipo las variables…?
  - siguiente, ultimo, p, q, r

Depende de la implementación

Nombres de expresiones de tipos II

- Redefinir noción de equivalencia
  - Equivalencia de nombres

La equivalencia de nombres considera cada nombre de un tipo como un tipo distinto, de modo que dos expresiones de tipo tienes equivalencia de nombres si y sólo si son idénticas

  - Equivalencia estructural

Con la equivalencia estructural, los nombres se sustituyen por las expresiones de tipos que definen. Así dos expresiones de tipos son estructuralmente equivalentes si y sólo si son idénticas una vez sustituidos los nombres por sus expresiones de tipo correspondientes

Ciclos en la representación de tipos I

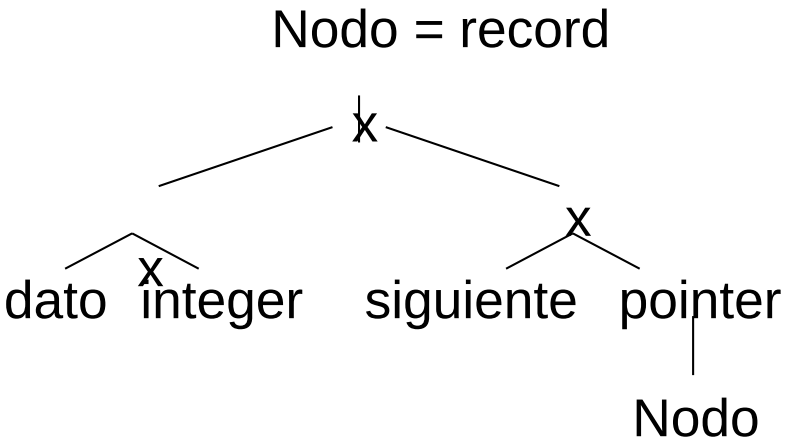
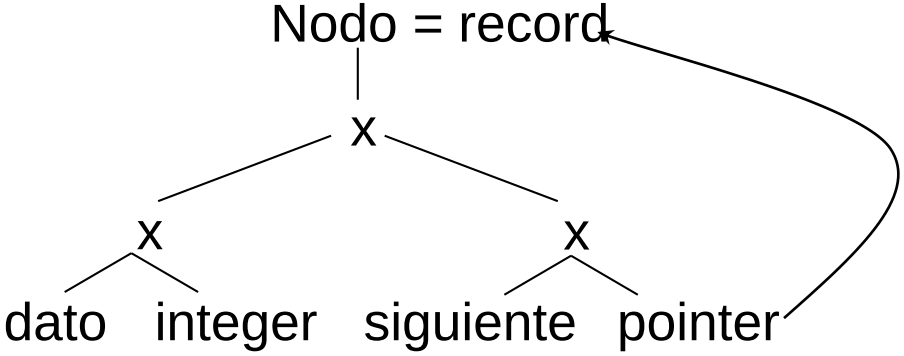
- Algunas estructuras tienen definición recursiva. Ej.: lista

```
Type enlace = ^nodo;
      nodo = record
                        dato : integer;
                        siguiente: enlace
                     end;
```

  - Nodo se define mediante enlace
  - Enlace la define mediante nodo

Ciclos en la representación de tipos II

- Representaciones recursivas
  - Representación no cíclica



  - Representación cíclica