7. Comprobaciones de tipos

1. Introducción.
2. Comprobaciones semánticas
3. Ámbito de declaraciones
4. Comprobaciones de tipo
5. Ejemplo de sistema de tipos.

v. 16

1. Introducción

Sistema de tipos

Sistema de tipo:

- Está formado por un conjunto de reglas que asignan expresiones de tipo a las construcciones de un lenguaje y que definen la equivalencia de tipos, la compatibilidad de tipos y la inferencia de tipos.

Comprobador de tipos:

- Implementación de un sistema de tipos.

Expresión de tipo

- Una expresión de tipo es un tipo básico o un constructor de tipos aplicado a una o más expresiones de tipo.
- El conjunto de tipos básicos y constructores de tipo depende del lenguaje fuente. Una **expresión de tipo** será:
  - *Un tipo básico: tentero, treal, tcar, tlogico, terror, y tvacio.*
  - *El nombre de una expresión de tipo.*
  - *Un constructor de tipos aplicado a expresiones de tipo:*
    - *tpuntero (T)*
    - *tvector (I_1xI_2x…xI_k, T)*
    - *T_1xT_2*
    - *D -> R*
    - *testructura((N_1 x T_1) x (N_2 x T_2) x … x (N_k x T_k))*

2. Comprobaciones semánticas

Comprobaciones dinámicas vs estáticas

- **Comprobación dinámica:**
  Se realiza durante la ejecución del programa objeto (en tiempo de ejecución).

- **Comprobación estática:**
  Se realiza en tiempo de compilación.

- Algunas comprobaciones de tipo solo pueden realizarse en tiempo de ejecución.

Comprobaciones estáticas

Para almacenar la información semántica de los objetos que aparecen en el programa fuente se utiliza la Tabla de Símbolos (TDS).

- *Comprobaciones del ámbito de las declaraciones*
- *Comprobaciones de tipo: Comprobación de que los tipos de los operandos y operadores de las expresiones son compatibles.*
- *Comprobación de declaración: Un identificador no puede usarse antes de ser declarado.*
- *Comprobación de unicidad: Los identificadores no pueden definirse más de una vez dentro del mismo bloque.*
- *Comprobación de parámetros: Los métodos (o funciones) deben invocarse con el número y tipo de parámetros adecuado,*
- *Otras: Comprobaciones de flujo de control, relaciones de herencia, unicidad de clases y métodos,…*

Comprobaciones dinámicas

- Dependen del contexto de la ejecución.

- *Verificar estado de la pila (stack) y montículo (heap)*
- *Verificación de desbordamientos (overflow y underflow)*
- *Divisiones por cero*
- *Verificaciones de direcciones e índices en variables indexadas*

…

- Un lenguaje es fuertemente-tipado si prohíbe, de forma que la implementación del lenguaje pueda asegurar su cumplimiento, la aplicación de una operación a cualquier objeto que no la soporte.
- Un lenguaje es estáticamente tipado, si es fuertemente tipado y las comprobaciones de tipo pueden realizarse en tiempo de compilación.
  - Pocos lenguajes son estáticamente tipados en el sentido riguroso. Pero se suele aceptar que lo son Ada (en su mayor parte), ó C
- Dinámicamente tipados: Lisp, Smalltalk, lenguajes de script,.. En general lenguajes con ámbito dinámico.

3. Ámbito de declaraciones

Ámbito de una variable

- Define el segmento de programa en el que la variable es accesible.
- Relaciona la declaración de la variable con su uso
- Lenguaje de ámbito estático (o léxico)

  Si el ámbito está completamente determinado por su posición en el código fuente.
- Lenguaje de ámbito dinámico

  Si el ámbito depende del estado durante la ejecución del programa.

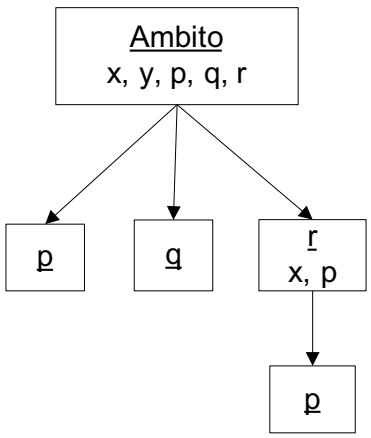
Ámbito estático vs dinámico

	función	Variables	Valor de y
Ámbito	Dinámico	Dinámico	6
	Estático	Dinámico	10
	Estático	Estático	5

```
programa ambito;
    var x, y : integer;
    funcion p;
        begin x := x * 2; end;
    funcion q;
        begin p() end;
    funcion r;
         var x: integer;
         funcion p;
              begin x:=x+1 end;
         begin x := 5; q(); y := x;
end;
    begin x := 0; r(); write(y);
end.
```

Ámbito estáticos (léxico) vs. ámbito dinámico

Relación estática

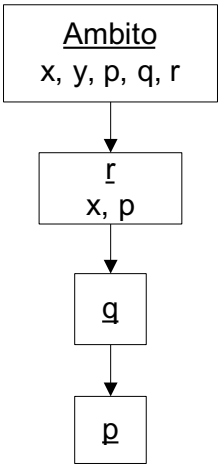


Ámbito estático vs dinámico

	función	Variables	Valor de y
Ámbito	Dinámico	Dinámico	6
	Estático	Dinámico	10
	Estático	Estático	5

```
programa ambito;
    var x, y : integer;
    funcion p;
        begin x := x * 2; end;
    funcion q;
        begin p() end;
    funcion r;
         var x: integer;
         funcion p;
              begin x:=x+1 end;
         begin x := 5; q(); y := x;
end;
    begin x := 0;  r(); write(y);
end.
```

Relación dinámica



Comprobaciones de ámbito

- Un mismo nombre puede identificar objetos distintos en distintas partes del programa, pero no se pueden solapar sus ámbitos.
- En lenguajes con ámbitos estáticos se permite anidamiento de declaraciones (C, C++, Java,…)
- Los método no necesitan estar declarados en la clase que lo utiliza si lo están en una antecesora.
- Los métodos pueden redefinirse

Ejemplo ámbito estático: Pascal

```
program uno ;
  var
    a, b : Tipo ;
    procedure pr1 (p1, p2 : Tipo);
        var
        c, d: Tipo ;
        function pr2 (p3, p4, p5: Tipo): Tipo;
            var e, c: Tipo ;
            begin … end ;
        procedure pr3 (p6, p7: Tipo);
            begin pr2 … end ;
        begin pr3 … end ;
    function pr4: Tipo;
      var f : Tipo ;
      begin …  end ;
  begin pr1 … end
```


4. Comprobaciones de tipos

Equivalencia de tipos

- En un lenguaje tipado estáticamente toda definición de un objeto (constante, variable, subrutina,…) debe especificar el tipo del objeto.

- ¿Cuándo son equivalente dos expresiones de tipo?
  - *Equivalencia por nombre:* Dos expresiones de tipo son equivalentes si tienen el mismo nombre.
  - *Equivalencia estructural:* Dos expresiones de tipo son equivalentes si representan expresiones estructuralmente equivalentes después de sustituir todos los nombres por las expresiones que representan.

Ejemplos

- Equivalencia estructural: Algol 68, Módula-3, C (casi todo), ML, Pascal (primeros compiladores)
- Equivalencia por nombre: Java, C#, Ada, Pascal estándar.

Compatibilidad de tipos

- No siempre se exige que 2 tipos sean iguales (equivalentes), puede bastar con que sean compatibles en el contexto en el que aparecen.

```
Ej.  int b;  float c, a;
       a = b + c ;
```

Conversiones de tipo

- Coerción:
  - Se permite un tipo en un contexto donde se esperaba otro.
  - La implementación del lenguaje debe convertir automáticamente al tipo esperado: Conversión de tipos implícita introducida por el compilador.
  - Puede requerir código para la comprobación de tipos en tiempo de ejecución.

- Conversión explícita:
  - Cuando el programador debe indicar explícitamente la conversión en el programa fuente.

Conversiones de tipo

- Sobrecarga
  - Un operador de función puede representar diferentes operaciones según el contexto en el que se usa.
  - La sobrecargar se resuelve determinando qué ocurrencia del símbolo sobrecargado se está empleando en cada contexto.

- Tipos de referencia genéricos
  - Se permiten objetos "contenedores" que apuntan a otros objetos
  - Ej. C y C++: void *

Inferencia de tipos

- Consiste en calcular (inferir) el tipo de un objeto o expresión.
- A veces no es sencillo
- El resultado de un operador aritmético suele tener el mismo tipo que los operadores
- Una asignación suele tener el tipo de la expresión de su lado izquierdo

5. Ejemplo de sistema de tipos

Declaraciones (1/6)

P → D D_F

T → **int** { *T.tipo = tentero* }

| **float** { *T.tipo = treal* }

| **bool** { *T.tipo = tlogico* }

| **struct** { C } { *T.tipo = testructura (C.tipo)* }

| T_1 * { *T.tipo = tpuntero(T_1.tipo)* }

D → D D

| ε

| T **id** ; { *InsertarTds (id.nom,"variable",T.tipo)* }

| T **id** L_I ; { *InsertarTds (id.nom,"variable",tvector(L_I.tipo,T.tipo)* }

D_F → T **id** (P_F) { *InsertarTds (id.nom,"funcion", P_F.tipo -> T.tipo)* }

{ D_1 L_Inst } D_F

| ε

Índices declaración array (2/6)

*/**** Índices de declaración de array ****/*

L_I → **[cte]** { *si (cte.tipo ≠ tentero)* ***OR*** *(cte.valor < 0)*

ent yyerror() ; L_I.tipo = terror ;

sino L_I.tipo = cte.valor }

| L_I_1 **[cte]** { *si (cte.tipo ≠ tentero)* ***OR*** *(cte.valor < 0)*

ent yyerror(); L_I.tipo = terror ;

sino si L_I_1.tipo == terror ent L_I.tipo = terror

sino L_I.tipo = L_I_1.tipo x cte.valor }

/**** Miembros de estructuras ****/

C → T **id** { *C.tipo = (id.nom x T.tipo)* }

| C_1 **;** T **id** { *C.tipo = C_1.tipo x (id.nom x T.tipo)* }

*/**** Parámetros formales ****/*

P_F → L_PF *{ P_F.tipo = L_PF.tipo }*

| ε *{ P_F.tipo = tvacio }*

L_PF → T **id** *{ InsertarTds (id.nom,"parametro",T.tipo) ;*
L_PF.tipo = T.tipo }

| L_PF$_1$ **,** T **id** *{ InsertarTds(id.nom,"parametro",T.tipo) }*
{ L_PF.tipo = L_PF$_1$.tipo x T.tipo }

Expresiones (4/6)

E → **cte** *{ E.tipo = cte.tipo }*

| **id** *{ Si* ***NOT*** *ObtenerTds(id.nom,* E.tipo*)*
ent E.tipo = terror ; yyerror() }

| **id** L_E *{ Si ObtenerTds(id.nom, tvector(*I,tipo*))* ***AND***
NumDimensiones(I) == L_E.ndim ***AND*** *(L_E.tipo ≠ terror)*
ent E.tipo = tipo sino E.tipo = terror ; yyerror() }

| **id** P_A *{ Si ObtenerTds(id.nom,"funcion",* D->R*)* ***AND*** *(D == P_A.tipo)*
ent E.tipo = R sino E.tipo = terror ; yyerror() }

| *** id** *{ Si ObtenerTds(id.nom, tpuntero(tipo)) ent E.tipo = tipo*
sino E.tipo = terror ; yyerror() }

| **& id** *{ Si ObtenerTds(id.nom, tipo) ent E.tipo = tpuntero(tipo)*
sino E.tipo = terror ; yyerror() }

| **id$_1$. id$_2$** *{Si ObtenerTds(id$_1$.nom, testructura(tipo))* ***AND***
BuscarCampo(tipo, id$_2$.nom, tipo-campo*)*
ent E.tipo = tipo-campo sino E,tipo = terror ; yyerror() }

(5/6)

/**** Índices de un array ****/

L_E → **[** E **]** { *Si E.tipo == tentero*

ent L_E.tipo = tentero sino L_E.tipo = terror ;

L_E.ndim = 1 }

L_E → L_E_1 **[** E **]** { *Si E.tipo == tentero* ***AND*** L_E_1*.tipo == tentero*

ent L_E.tipo = tentero sino L_E.tipo = terror;

$L\_E.ndim = L\_E_1.ndim + 1$ }

/**** *Parámetros actuales* ****/

P_A → ε { P_A.tipo : tvacio }

| **(** L_PA **)** { P_A.tipo = L_PA.tipo }

L_PA → E { L_PA.tipo = E.tipo }

| L_PA_1 **,** E { $L\_PA.tipo = L\_PA_1.tipo \times E.tipo$) }

(6/6)

/**** Algunas instrucciones ****/

I → id **=** E **;** { *Si* ***NOT*** *ObtenerTds(id.nom, id.tipo)* ***OR*** *id.tipo* ≠ *E.tipo*

ent yyerror(); I.tipo = terror

else I.tipo = id.tipo}

| while (E) I { *Si E.tipo* ≠ *tlogico ent yyerror(); I.tipo = terror*

else I.tipo = tvacio}

| { L_Inst **}**

L_Inst → L_Inst I

| ε

Ejercicio

S → **id** := E

| S ; S

| **for** (S ; E ; S) S

E → E ≺ E

| E # E

| **id**

ETDS que realice la comprobación de tipos. Los operadores ≺ y # son sobrecargados.

≺ Operador de orden que puede tener operandos enteros y booleanos.

Suma de enteros, el 'o' lógico o la concatenación de cadenas de caracteres, según los operandos sean enteros, booleanos o cadenas respectivamente.

La tabla de símbolos se supone iniciada con los tipos de cada identificador.

En ' for (S1 ; E ; S2) S3 ' las ocurrencias del terminal S1 y de S2, deben contener alguna instrucción de asignación y el no terminal E debe reescribirse de forma que contenga algún identificador que ocurra en la parte izquierda de alguna asignación que aparezca en S1 o en S2.

Solución

S → **id** := E	S.ident := {id.nom} Si BuscaTipo(id.ind) <> E.tipo ent MemError();
| S_1 , S_2	S.ident := S_1.ident ∪ S_2.ident ;
| **for** (S_1 ; E ; S_2) S	Si S_1.ident=∅ or S_2.ident=∅ ent MemError() sino si E.ident ∩ (S_1.ident ∪ S_2.ident) = ∅ ent MemError() sino si E.tipo <> Tlogico ent MemError() ; S.ident := ∅ ;
E → E_1 ≺ E_2	Si E_1.tipo <> E_2.tipo or (not E_1.tipo in [Tentero, Tlogico]) ent MemError() ; E.tipo := Terror sino E.tipo := E_1.tipo E.ident := E_1.ident ∪ E_2.ident ;
| E_1 ⊕ E_2	Si E_1.tipo <> E_2.tipo or (not E_1.tipo in [Tentero, Tlogico, Tcadena]) ent MemError() ; E.tipo := Terror sino E.tipo := E_1.tipo ; E.ident := E_1.ident ∪ E_2.ident ;
| **id**	E.tipo := BuscaTipo (id.ind) ; E.ident := {id.nom}