TEMA 7 VERIFICACIÓN DE TIPOS

Francisco José Ribadas Pena

PROCESADORES DE LENGUAJES
4º Informática
ribadas@uvigo.es

13 de marzo de 2007

7.1 Introducción

Tarea clave del Análisis Semántico, garantiza que el programa "tiene sentido".

OBJETIVO: Asegurar que el tipo de una construcción del programa coincida con el previsto en sus contexto.

- Compatibilidad de tipos en las asignaciones
- Operadores aplicados a datos del tipo adecuado
- Nº y tipo correcto en los parámetros de funciones

Además, se debe considerar:

- CONVERSIÓN DE TIPOS (cuando estos sean compatibles)
  - Conversión IMPLÍCITA (coherciones) $\longrightarrow$ realizada por el compilador
  - Conversión EXPLÍCITA $\longrightarrow$ realizada por el programador
- SOBRECARGA DE OPERADORES y FUNCIONES POLIMÓRFICAS
  - Símbolos del lenguaje que tienen asociados distintos tipos y operaciones en función del contexto.
  - Ejemplo: Operador "+": $\begin{cases} \text{suma de enteros} \\ \text{suma de reales} \\ \text{concatenación de cadenas} \end{cases}$

7.2 Sistemas de Tipos

Sistema de Tipos: Conjunto de reglas que permiten asociar tipos a las construcciones del lenguaje y verificar su corrección.

- Determinan el conj. de expresiones de tipo admisibles en el lenguaje

Expresiones de tipo: Tipos asociados a las construcciones del lenguaje.

- Una *expresión de tipo* es o bien un tipo básico, o es el resultado de aplicar un constructor de tipo a otra expresión de tipo de acuerdo a unas reglas de construcción (dadas por el lenguaje).
  - Tipos básicos y constructores de tipo dependen del lenguaje
  - Expresiones de tipo pueden tener asociados nombres

Tipos Básicos: `char, boolean, real, integer`

- Tipos especiales:
  - `error`: Indica que no se puede asociar una expr. de tipo correcta
  - `void`: Indica que una construcción del lenguaje es correcta, pero no tiene ningún tipo asociado.
    - Útil en comprobación de sentencias

Constructores de tipo: Permiten formar tipos complejos a partir de otros más simples.

- MATRICES: Siendo T una expr. de tipo , "$array(I, T)$" es el tipo de una matriz con elementos de tipo T e índices del tipo I.
  - $T \rightarrow$ tipo de los elementos del array
  - $I \rightarrow$ tipo de los índices (generalmente un rango)

  Ejemplo:

PASCAL:	`A: array [0..9] of integer;`	$array(0.,9, integer)$
C:	int A[10];	

- PRODUCTOS: Si T_1 y T_2 son expr. de tipo, "$T_1 \times T_2$" es una expr. de tipo. ($\rightarrow$ se supone "$\times$" asociativo por la izq.)

- REGISTROS: Similar a productos, pero con nombre asociados a los campos
  - Si $u_1, u_2, \ldots, u_n$ son los campos de tipos $T_1, T_2, \ldots, T_n$, la expr. de tipo asociada al registro es:
    "$record(u_1 : T_1 \times u_2 : T_2 \times \ldots \times u_n : T_n)$"

  Ejemplo:

```
struct {                    record
   char nombre[10];            nombre: array [0..9] of character;
   float altura;               altura:real;
}                           end;
```

$record(nombre : array(0.,9, char) \times altura : real)$

- PUNTEROS: Si T es una expr. de tipo, "$pointer(T)$" representa al tipo de dato que tiene como valores el conjunto de posiciones de memoria que pueden almacenar datos del tipo T.
  Ejemplo:

`char * p;` `p:^character`	$pointer(char)$

- FUNCIONES: Siendo D la expr. de tipo de los argumentos de la función y T el tipo del valor devuelto, la expr. de tipo de la función será: "$D \rightarrow T$"
  - Para funciones con múltiples argumentos se usarán productos

  Ejemplo:

`float * f(char a,b);` `function f(a,b:character):^real`	$(char \times char) \rightarrow pointer(real)$

- Otros constructores: conjuntos, pilas, colas, ficheros, etc,...
- NOTA: Dentro de las expresiones de tipo podrán incluirse variables
  - Representa un componente que puede tener cualquier expr. de tipo asociada o del cual se desconoce su tipo.
  - Util para manejar funciones polimiórficas.

7.3 Declaración y Comprobación de Tipos

Varemos un ejemplo sencillo del manejo de declaraciones de tipo y de comprobación de tipos en expresiones, funciones y sentencias.

Se usará un lenguaje simplificado:

- 4 tipos básicos
  - char
  - boolean
  - integer
  - real
- 3 constructores
  - matrices
  - puntero
  - funciones (con 1 argumento)

Gramática simplificada (no útil para A. Sintáctico)

- ATRIBUTOS:tipo: expr. de tipo asociada a un nombre/fragmento de código

(a) DECLARACIÓN DE TIPOS

$Decl \rightarrow Decl\ ;\ Decl\ ;$	
$Decl \rightarrow \varepsilon$	
$Decl \rightarrow id\ :\ Tipo$	{ TDS_insertar(id.texto, Tipo.tipo)}
$Tipo \rightarrow char$	{ Tipo.tipo = $char$}
$Tipo \rightarrow boolean$	{ Tipo.tipo = $boolean$}
$Tipo \rightarrow integer$	{ Tipo.tipo = int}
$Tipo \rightarrow real$	{ Tipo.tipo = $real$}
$Tipo \rightarrow \wedge Tipo$	{ Tipo_0.tipo = $pointer$(Tipo_1.tipo}
$Tipo \rightarrow array\ [num]\ of\ Tipo$	{ Tipo_0.tipo = $array$(1..num.valor, Tipo_1.tipo}

(b) COMPROB. DE TIPOS EN EXPRESIONES

$Exp \rightarrow num\_int$ { Exp.tipo = integer}
$Exp \rightarrow num\_real$ { Exp.tipo = real}
$Exp \rightarrow literal$ { Exp.tipo = char}
$Exp \rightarrow id$ { Exp.tipo = TDS_obtenerTipo(id.texto)}

Operadores aritméticos y lógicos

$Exp \rightarrow Exp\ mod\ Exp$ { Exp_0.tipo = if (Exp_1.tipo=$integer$) and (Exp_2.tipo=$integer$) then $integer$ else $error$ }

$Exp \rightarrow Exp == Exp$ { Exp_0.tipo = if (Exp_1.tipo=$integer$) and (Exp_2.tipo=$integer$) then $boolean$ else $error$ }

Acceso a matrices y punteros

$Exp \rightarrow Exp\ [\ Exp\ ]$ { Exp_0.tipo = if (Exp_2.tipo=$integer$) and (Exp_1.tipo=$array(S,T)$) then T else $error$ }

$Exp \rightarrow \wedge Exp$ { Exp_0.tipo = if (Exp_1.tipo=$pointer(T)$) then T else $error$ }

(c) COMPROB. DE TIPOS EN LLAMADAS A FUNCIÓN

$Exp \rightarrow id(\ Exp\ )$ { tipoID = TDS_obtenerTipo(id.texto)
Exp$_0$.tipo = if (tipoID=$S \rightarrow T$) and
(Exp$_1$.tipo=S)
then T
else $error$ }

NOTA: Con funciones de más de un argumentos se deberían usar productos.

(d) COMPROB. DE TIPOS EN SENTENCIAS

NOTA: A las sentencias se les asociará el tipo $void$ si los tipos de las expresiones que contienen son correctos.

- Consideraremos: asignaciones, if simple, bucle while y secuencias.

$Sent \rightarrow id := Exp$ {tipoID = TDS_obtenerTipo(id.texto)
Sent.tipo = if (tipoID=Exp.tipo)
then $void$
else $error$ }

$Sent \rightarrow if\ Exp\ then\ Sent$ {Sent$_0$.tipo = if (Exp.tipo=boolean)
then Sent$_1$.tipo
else $error$ }

$Sent \rightarrow while\ Exp\ do\ Sent$ {Sent$_0$.tipo = if (Exp.tipo=boolean)
then Sent$_1$.tipo
else $error$ }

$Sent \rightarrow Sent\ ;\ Sent\ ;$ {Sent$_0$.tipo = if (Sent$_1$.tipo=void) and
(Sent$_2$.tipo=void)
then $void$
else $error$ }

7.4 Equivalencia de Tipos

Objetivo: Determinar si dos expr. de tipo se consideran equivalentes.

Dos modos de equivalencia de tipos:

1. **Equivalencia estructural:**
   - Dos expr. de tipo son equivalentes estructuralmente si son el mismo tipo básico, o si están formadas por la aplicación del mismo constructor de tipos a 2 tipos estructuralmente equivalentes.
   - Ejemplo: (Pascal)

```
type
  t_vector: array [0..9] of integer;
  t_matriz : array [0..9] of array [0..9] of integer;
var
  uno:    array [0..9] of integer;
  dos:    array [0..9, 0..9] of integer;
  tres:   t_matriz;
  cuatro: array [0..9] of t_vector;
  cinco:  array [0..9] of t_vector;
```

     - Los tipos de $\{dos, tres, cuatro, cinco\}$ son estructuralmente equiv., su expr. de tipo es $array(0.,9, array(0.,9, integer))$.
     - El tipo de uno no es estructuralmente equivalente con ninguna de esas variables, pero si con $cuatro[4]$ (expr. tipo: $array(0.,9, integer)$).
2. **Equivalencia nominal:**
   - Dos expr. de tipo son nominalmente equivalentes si son el mismo tipo básico o si a ambas se les ha asociado el mismo nombre.
     - Definición de eq. nominal depende de como implemente el lenguaje el manejo de tipos.
     - Algunos lenguajes asignan nombre implícitos para cada aplicación de un constructor de tipo.
   - Ejemplo:
     - En el ej. anterior, sólo $cuatro$ y $cinco$ son nominalmente equiv. Su expr. de tipo es $array(0.,9, t\_vector)$.
     - Los dos no son nominalmente equiv. con dos, su expr. de tipo es $array(0.,9, (array(0.,9, integer))$

- Otro ejemplo:

```
type
  T0 : integer;  T1: record            T2: record
                       a, b: integer;       x: integer;
                       c: T0;               y,z: T0;
                     end;                 end;
```

  - Con eq. estructural: $T1$ y $T2$ son equivalentes.
  - Con eq. de nombres: No son equivalentes.
    - c, y, z son del mismo tipo
    - a, c son de tipos distintos

(a) Implementación de la equivalencia estructural

IDEA: Recorrido recursivo de las expr. de tipo

FUNCTION esEquivalente(S, T): boolean

```
begin
  if S and T son el mismo tipo básico
  then return TRUE;
  else if (S = array(S1, S2)) and (T = array(T1, T2))
    then return [esEquivalente(S1,T1) and esEquivalente(S2,T2)];
  else if (S = S1 × S2) and (T = T1 × T2)
    then return [esEquivalente(S1,T1) and esEquivalente(S2,T2)];
  else if (S = pointer(S1)) and (T = pointer(T1))
    then return [esEquivalente(S1,T1)];
  else if (S = S1 → S2) and (T = T1 → T2)
    then return [esEquivalente(S1,T1) and esEquivalente(S2,T2)]
  else return FALSE;
end
```

NOTA: La equiv. nominal es mas sencilla de implementar

- Basta comprobar que coinciden los nombres asociados las expr. de tipo.
- PROBLEMA: Es menos flexible para el programador.

(b) Representación de expresiones de tipo

Estructuras de datos internas usadas por el compilador para manejar las expr. de tipo.

- Deseable representaciones manejables que permitan comprobación de tipos eficiente
- Dependen del tipo de equivalencia de tipos considerada

Dos posibilidades: { codificación binaria / estructuras arbóreas

1. CODIFICACIÓN
   - Usada para expr. de tipo sencillas
   - Se asigna un número binario a cada expr. de tipo posible
   - Comparación de tipos muy eficiente
     - Uso de operaciones a nivel de bits (desplazamientos, AND, OR, uso máscaras, etc)

   Ejemplo: $array(pointer(char)) \rightarrow$ 101 100 001

TIPO BÁSICO	CODIFIC
boolean	000
char	001
integer	010
real	011

CONSTRUCTOR	CODIFIC
pointer	100
array	101
funcion	110

2. ESTRUCTURAS ARBÓREAS
   - Expr. de tipo representadas en forma de árboles
     - hojas $\rightarrow$ tipos básicos
     - nodos internos $\rightarrow$ constructores de tipo
   - Método más general que anterior
   - Comparación tipos supone recorrido sobre el árbol
   - Ejemplo: $(real \times (char \times char)) \rightarrow pointer(integer)$

   - Otra posibilidad: Uso de DAG (grafos dirigidos acíclicos)
     - Forma condensada de representar árboles
     - Estructuras repetidas se representan implícitamente
     - Representación compacta $\Rightarrow$ { menos memoria; comparación más rápida; construcción más compleja }

7.5 Conversión de Tipos

Conversión de tipos: Modificación del tipo asignado a una construcción del programa.

- Dos tipos $\left\{ \begin{array}{l} \text{implícita: realizada por el compilador automáticamente} \\ \text{explícita: realizada por programador (cast)} \end{array} \right.$.

Conversión implícita:

- Se limita a situaciones donde no se pierde información.
  - Conversión de tipo más específico a otro más general
  - Ej.: cambio de entero a real, de char a int (en C), etc,..
- Relacionado con el uso de operadores sobrecargados
- Supondrá la inclusión de nuevo código adicional para realizar la conversión en el código objeto generado.
- Ejemplo: Conversión implícita con operadores aritméticos

$Exp \rightarrow num\_entero$ {Exp.tipo = integer }

$Exp \rightarrow num\_real$ {Exp.tipo = real }

$Exp \rightarrow id$ {Exp.tipo = TDS_obtenerTipo(id.texto)}

$Exp \rightarrow Exp\ \mathbf{op}\ Exp$ {Exp_0.tipo =
if (Exp_1.tipo=$integer$) and (Exp_2.tipo=$integer$)
then return ($integer$)
else if (Exp_1.tipo=$integer$) and (Exp_2.tipo=$real$)
then convertirReal(Exp_1)
return($real$)
else if (Exp_1.tipo=$real$) and (Exp_2.tipo=$integer$)
then convertirReal(Exp_2)
return($real$)
else if (Exp_1.tipo=$real$) and (Exp_2.tipo=$real$)
then return($real$)
else return($error$) }

Siendo **op** un operador aritmético: $+, -, *, /$.