Principios de Lenguajes de Programación

Equivalencia de tipos

Facultad de Informática
Universidad Nacional del Comahue

Primer Cuatrimestre

Índice

- Sistema de Tipo
- Chequeo de Tipo
- Equivalencia de tipos (Compatibilidad)

Chequeo de Tipo de Dato

- Verificación de tipo (estática o dinámica)
- ¿Qué significa decir que dos tipos son **“los mismos”**?
- ¿Qué significa decir que dos objetos de datos del mismo tipo son “**iguales**”?

Chequeo de Tipo de Dato

- Igualdad de tipos

```
Program principal(entrada, salida);
Type Vec1: array[1..10] of real;
     Vec2: array[1..10] of real;
Var X,Z: Vec1;
      Y: Vec2;
 …
Begin
...
    X:=Y;
    X:=Z;
End
```

Chequeo de Tipo de Dato

- ***Chequeo de Tipo***: actividad que asegura que los operandos de un operador (parámetros/función) son de tipos compatibles
- ***Tipo Compatible***: operando válido para el operador o se permite conversión (implícita) del tipo. Ej `int X real Y,Z   Z=X+Y`
- ***Error de tipo***: es la aplicación de un operador a un operando de un tipo inapropiado

Chequeo de Tipo de Dato

- Ligaduras de variables a tipo son estáticas entonces el chequeo es estático
- Ligaduras de variables dinámicos requiere chequeo en tipo de ejecución (chequeo dinámico)
- Lenguaje fuertemente tipado: errores de tipo siempre detectado

Sistema de tipos

Sistema de tipos: Conjunto de reglas que estructuran y organizan una colección de tipos.

- El objetivo del sistema de tipos es lograr que los programas sean tan seguros como sea posible.

Chequeo de Tipo de Dato

- Un ***sistema de tipo de dato*** tiene reglas para:
  - *Equivalencia de tipo* (cuándo los tipos de datos valores son iguales)
  - *Compatibilidad de tipo* (cuando un valor del tipo A puede usarse en un contexto en el cual se esperaba un tipo B)
  - *Inferencia de tipo* (cuál es el tipo de una expresión, calculado a partir de los tipos de los operandos)

Sistema de tipos: Especificación

- Tipo y tiempo de chequeo
- Reglas de equivalencia y conversión
- Reglas de inferencia de tipo
- Nivel de polimorfismo del lenguaje

Tipo y tiempo de chequeo

- El chequeo de tipo, consiste en verificar que el tipo de una entidad corresponda al esperado por el contexto.
- Si no coinciden, puede tratarse de un error o se aplican reglas de coerción o reglas de equivalencia.

Sistemas de Tipos: Lenguajes fuertemente tipados

- Un sistema de tipo puede conducir a un LP fuertemente tipado si:
  - Solo se puede utilizar tipos predefinidos
  - Todas las variables son declaradas y asociadas a un tipo específico
  - Todas las operaciones son especificadas determinando con exactitud su signatura (tipos requeridos por sus op y e tipo de resultado)
- Especificación necesita de reglas para la
  - Conversión o coerción
  - Equivalencia o compatibilidad

Chequeo de Tipo de Dato

- Compatibilidad / Equivalencia de Tipo
  - Compatibilidad es mucho más útil, nos dice que se PUEDE hacer
    - Las reglas de compatibilidad dicta el tipo de operandos que son aceptables para cada una de las operaciones (ve errores)
  - Muchas veces se usan en forma intercambiable (incorrectamente)

Sistema de tipos: Reglas de conversión

- Un sistema de tipos flexible brinda **reglas de conversión** que permiten aceptar datos de un tipo Q en un contexto en el cual se espera un dato de tipo T
- La conversión puede ser:
  - Con pérdida (limitante) insegura
  - Sin pérdida (expansora) segura

  Sea **f: T1 → R1**
- La operación **f** puede ser invocada con un argumento de tipo **T2** , si el lenguaje brinda una regla que permita convertir los valores de tipo **T2** en valores de tipo **T1**
- La operación **f** puede ser invocada en un contexto en el se espera un valor de **R2** , si el lenguaje brinda reglas para convertir valores de tipo **R1** en valores de tipo **R2**

Sistema de tipos: Reglas de conversión

- En la mayoría de los lenguajes durante la asignación hay una operación implícita de dereferenciamiento

$$x := x + 1$$

Las reglas de conversión en los lenguajes:

- Fortan es un lenguaje que resuelve las conversiones de acuerdo a
- una jerarquía de tipos:
- COMPLEX > DOUBLE PRECISION > REAL > INTEGER
- Ada exige que todas las conversiones se hagan explícitamente
- En Pascal las conversiones no están especificadas con precisión y dependen de la implementación

Sistema de tipos: Reglas de conversión

- En C

```
int x,y
real z
x:= y + z
```

y se convierte a **real** antes de evaluar la suma y luego el resultado se convierte a **int**

- El programador puede utilizar el casting para indicar explícitamente la conversión

```
x:= y + (int) z
```

- Java tiene un tratamiento diferenciado para tipos primitivos y tipo clase

Compatibilidad de Tipo

- Trabajando sobre objetos de datos de tipo de dato estructurado (simplifica el análisis)
- La equivalencia de tipo es una forma más estricta de compatibilidad de tipo.
- Clasificador por Niveles de Restricciones
  - Nombre
  - Declaración
  - Estructura

Compatibilidad de Tipo: Nombre

- Compatibilidad de Tipo por Nombre: dos variables (objetos de datos) son de tipo compatible si en la declaración utilizan el mismo nombre de tipo.
- No puede haber tipo anónimos.

```
Var W:array[1..10] of real
```

- W tipo sin nombre. No como argumento subprog.
  - Fácil de implementar
  - Altamente restrictivo

Compatibilidad de Tipo

- Compatibilidad de Tipo por Nombre: ejemplo
  - Los subrangos (de enteros) no son compatibles con los enteros: ADA equivalencia de nombre estricto

```
type Indextype is 1..100;
count : Integer;
index : Indextype;
```

    - Count y Index NO son compatibles por nombre
  - Los parámetros formales deben tener el mismo nombre de tipo que los parámetros reales (pe Pascal)
    - Los tipos deben definirse globales
    - Los subprogramas no pueden establecer tipos locales

Compatibilidad de Tipo: Declaración

- Compatibilidad de Tipo por Declaración: dos variables (objetos de datos) son de tipo compatible si utilizan la misma declaración de tipo (ie, si conducen al a misma expresion de tipo original luego de una serie de re-delcaraciones,)
  - Incluye a Compatibilidad por Nombre
  - Fuertemente restrictivo

Compatibilidad de Tipo: Declaración

- Ejemplo

```
type
   T1, T2 = array[1..10] of real;
   T3 = array[1..10] of real;
var V11, V12: T1;
    V2: T2;
    V3: T3;
    V4, V5 : array[1..10] of real;
```

- Compatible por Nombre:
  - T1 y T2
  - V11 y V12
  - V4 y V5
- Compatible por Declaración
  - T1, T2 y T3
  - Compatibles por Nombre

Compatibilidad de Tipo

- Tipos Anónimos
  - Permitidos en algunos lenguajes (ej Ada)
  - Todos son único, no son compatibles

  - Compatibles en la declaración del tipo
    - Type List_10 is array [1..10] of Integer;
    - C, D: List_10;

Compatibilidad de Tipo: Estructura

- Compatibilidad de Tipo de Dato por Estructura significa que dos variables (objetos de datos) son de tipo compatible si sus tipos tienen idénticas estructuras
  - Más flexible, pero más difícil de implementar
  - Se debe comparar la estructura completa de dos tipos
- Existen definiciones más relajadas:
  - … “similar estructuras” ...

Compatibilidad de Tipo

- Problemas:
  - ¿Dos tipos estructurados son equivalentes si la estructura se autoreferencia (lista vinculada)?
  - ¿Dos tipos registros son compatibles si son estructuralmente iguales pero con:
    - distintos nombres de campos?
    - Iguales nombres pero en otro orden?

Compatiblidad de tipo

- Ejemplo:

```
struct Rec1          struct Rec2          struct Rec3
{ char x;            { char x;            { int y;
  int  y;              int  y;              char x;
  char z[10]           char z[10];          char z[10];
};                   };                   };
```

- Si m1 es tipo Rec1, m2 es Rec2 y m3 es Rec3
- Es
  - ¿m1=m2? ¿m1=m3? ¿m2=m3?
- ¿Dos arreglos son compatibles si tienen la misma estructura pero difieren en los subíndices
    - ej, [1..10] and [0..9]

Compatibilidad de Tipo

- Dos tipos estructurados compatibles por estructura
  - No se pueden diferenciar los tipos (por ejemplo, dos unidades de medida diferente, ambas float)

```
type celsius = Float;

type fahrenheit = Float;
```

Compatibilidad de Tipo

- Uso de lenguajes
  - Pascal: normalmente por estructura, pero a veces por nombre (parámetros formales)
  - C: estructura, excepto en las *structs y union*
    - Cada struct y union crea un nuevo tipo, que son incompatibles con otros tipos
    - Typedef, no define un nuevo tipo, lo renombra
      - Compatible con el tipo original
  - C++: por nombre

Equivalencia de Tipo de Dato

- Sintaxis Java:
  `class A { int x; double y;}` and
  `class B { int x; double y;}`
  los conjuntos que representan `A` y `B` son los mismos:

```
A a = new B();
```

- Java usa
  - *equivalencia por nombre* para clases e interfases,
  - *equivalenica por estructura* para arreglos

Equivalencia de Tipo de Dato

- Pascal

```
type
    IntPtr = ^integer;
    Age = integer;
var
    x: IntPtr;
    y: ^integer;
    i: Age;
    a: integer;
```

- x, y no son equivalentes
- a, i si son equivalentes

- Ejemplos en el libro: Programming Languages Louden 3ed pp.352

Bibliografía

- Pratt, Terrance W., Programming Languages: Design and Implementation, Cap. 6 (4 ed)
- Sebesta, Robert, Concepts of Programming Languages, 9th Edition. 2009. Cap. 6
- Material de apoyo en pedco.