Lenguajes de Programación

Tipos de dato

Ma. Laura Cobo

Universidad Nacional del Sur
Departamento de Ciencias e Ingeniería de la Computación
2018

La noción de tipo

La mayoría de los lenguajes incluye esta noción vinculada a expresiones y objetos de dato. Los principales razones:

- Proveen un contexto implícito para muchas operaciones, de esta manera el programador no debe explicitar el contexto.
- Limitan el conjunto de operaciones que pueden realizarse en un programa semánticamente válido.

2

Generalidades

Abstracción de datos en los lenguajes de programación

Trata con las componentes de un programa que son sujeto de computación. Esta basada en las propiedades de los objetos de dato y las operaciones de dichos objetos.

- Tipos de datos
- Sistemas de tipos
- Encapsulamiento y abstracción

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.

3

Sistemas de Tipos

Informalmente consiste de:

- Un mecanismo para definir tipos y asociarlos con ciertos constructores del lenguaje
- Un conjunto de reglas para equivalencias de tipos, compatibilidad, inferencia, etc.

Los constructores involucrados son aquellos que manejan valores o hacen referencias a objetos que tienen valores.

4

Chequeo de tipos

Es el proceso a través del cual se garantiza que un programa obedece las reglas de compatibilidad que el lenguaje de programación define.

Un lenguaje es "*estáticamente tipado*" si su tipado es fuerte y el chequeo puede hacerse en compilación.

En un sentido estricto muy pocos lenguajes entran en esta categoría.

Lo habitual es que realicen todas las tareas de chequeo posibles en compilación y el resto se realiza en ejecución.

5

Tipado dinámico

Gano popularidad con los lenguajes de scripting.

Responde a la pregunta:

¿vale la pena chequear todo lo que podemos en tiempo de compilación?

6

Clasificación de tipos

Varia de lenguaje en lenguaje.

La mayoría provee tipos pre-definidos y constructores de tipo

Entre los predefinidos se encuentran:

- Tipos numéricos
- Caracteres
- Booleanos

Constructores básicos:

- Enumerados
- subrangos

Tipos compuestos:

- Registros
- Uniones (registros variantes)
- Arreglos
- Conjuntos
- Punteros
- Listas
- archivos

7

Sistemas de Tipos

Con frecuencia la **seguridad** compromete la **flexibilidad** del lenguaje

Cual de las dos características termina siendo la más preponderante o buscada por el sistema depende de a que tipo de desarrollos se apunte con el lenguaje de programación

Los lenguajes orientados al desarrollo de aplicaciones complejas buscan mayor seguridad y se caracterizan por tener un sistema de tipos estricto

Los lenguajes script orientados al desarrollo rápido, buscan mayor flexibilidad de modo que se caracterizan por tener un sistema de tipos relajado

8

Sistemas de Tipos: dureza del tipado

El **sistema de tipos** determina el grado de **dureza** respecto a una característica, cuando especifica restricciones sobre como se incorpora o manipula esa característica en el lenguaje.

Ej. Grado de dureza del sistema tipos con respecto a las expresiones mixtas

Tipado menos duro	Tipado más duro
A = 2 B= "2" Concatenar(A,B) {retorna "22"} Sumar(A,B) {retorna 4}	A = 2 B= "2" Concatenar(A,B) {error de tipos} Sumar(A,B) {error de tipos} Concatenar(srt(A),B) {retorna "22"} Sumar(A,int(B)) {retorna 4}

9

Sistemas de Tipos: Especificación

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

Para cada ítem se puede determinar el grado de dureza del sistema de tipos

Observación Importante: cada ítem puede involucrar varias características que se pueden evaluar en forma independiente.

Tipo y tiempo de chequeo

- El chequeo de tipo, consiste en verificar que el tipo de una entidad corresponda a 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: errores de tipo

- Errores en el uso del lenguaje: Errores sintácticos o semánticos
- Errores en la aplicación: desviaciones de la aplicación con respecto a la especificación

Ej. Un acceso ilegal que al pasar inadvertido provoca un error de **aplicación**

Ej. Un error chequeado y notificado por el **lenguaje** que permite su depuración

- Idealmente en el sistema de tipos se establecen restricciones que aseguran que no se van a producir errores de tipos en la **aplicación**, gracias a que ya han sido detectados en el **lenguaje**

11

Sistemas de Tipos: tipo y tiempo de chequeo

Tipos de ligadura

- Tipado estático: ligaduras en compilación
- Tipado dinámico: ligaduras en ejecución, no es seguro
- Tipado seguro: no es estático, ni inseguro

12

Sistemas de Tipos: Lenguajes fuertemente tipados

Definición 1: Un lenguaje se dice **fuertemente tipado (type safety)** si el sistema de tipos impone restricciones que aseguran que no se producirán errores de tipo en ejecución

Definición 2: Un lenguaje se dice **fuertemente tipado (type safety)** si todos los errores de tipo se detectan

En esta concepción, la intención es evitar los errores de **aplicación** y son tolerados los errores del **lenguaje**, detectados tan pronto como sea posible

13

Sistemas de Tipos: Lenguajes fuertemente tipados

Un sistema de tipos puede conducir a un LP fuertemente tipado requiriendo que:

- Solo se pueden utilizar tipos predefinidos
- Todas las variables son declaradas y asociadas a un tipo específico
- Todas las operaciones son especificadas determinando con exactitud su signatura (los tipos requeridos por sus operandos y el tipo del resultado)

Haciéndose eco de la segunda definición, se puede alcanzar a la posición de fuertemente tipado, con requerimientos leves. Es decir especificando con precisión reglas para la

- Conversión o coerción
- Equivalencia o compatibilidad

14

Sistemas de Tipos: Reglas de conversión

Un sistema de tipos flexible brinda **reglas de conversión** o **coerción** 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: T_1 → R_1**

La operación **f** puede ser invocada con un argumento de tipo **T_2**, si el lenguaje brinda una regla que permita convertir los valores de tipo **T_2** en valores de tipo **T_1**

La operación **f** puede ser invocada en un contexto en el se espera un valor de **R_2**, si el lenguaje brinda reglas para convertir valores de tipo **R_1** en valores de tipo **R_2**

15

Sistemas 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

16

Sistemas de Tipos: Reglas de conversión en los LP

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
```

C++ es similar a **C**

Java tiene un tratamiento diferenciado para tipos primitivos y tipo clase

17

Sistemas de Tipos: Reglas de equivalencia

Dos tipos de dato son equivalentes si cada uno de ellos puede aparecer en un contexto en el que opera una aparición del otro.

NO HAY CONVERSIÓN

En un lenguaje fuertemente tipado las reglas equivalencia deben quedar establecidas en el diseño del mismo.

18

Sistemas de Tipos: criterios de equivalencia

Equivalencia por nombre: dos variables son de tipos equivalentes si han sido declaradas en una misma instrucción o con exactamente el mismo nombre de tipo

Equivalencia por declaración: dos variables son equivalentes si conducen a la misma expresión de tipo original luego de una serie de re-declaraciones

Equivalencia por estructura: dos tipos de datos son equivalentes si están definidos por dos expresiones de tipo idénticas.

19

Sistemas de Tipos: equivalencia por estructura

A la hora de implementar la equivalencia por estructura es necesario plantear como resolver ciertos problemas:

1. el problema de la circularidad
2. Determinar si dos registros con diferentes nombres para sus campos son compatibles
3. Determinar si dos arreglos con la misma cardinalidad y rango son equivalentes.

Consideraciones

- La compatibilidad por nombre es mas simple de implementar
- La compatibilidad por nombre también es mas fuerte que la compatibilidad por estructura

20

Sistemas de Tipos: criterios de equivalencia

```
type
    T1: array[1..10] of int
    T2: array[1..10] of int
    T3: T2

var
    v1, v2: array [1..10] of int
    v3, v4: T1
        v5: T2
        v6: T2
        v7: T3
```

21

Sistemas de Tipos: criterios de equivalencia en LP

En **Ada**

```
Type Derivado_INT is new integer range 1..100;
Subtype Subrango_INT is integer range 1..100;
```

Adopta compatibilidad por nombre. Debido a que incorpora subtipos; define que las variables que pertenezcan a diferentes subtipos del mismo tipo base, son compatibles.

En **C**

Equivalencia por estructura salvo para registros y unión que tienen equivalencia por declaración

En **Java**

Para las clases la equivalencia esta dada por la jerarquía de herencia

22

Sistemas de Tipos: Inferencia de tipos

La **inferencia de tipos** permite que el tipo de una entidad declarada se "infiera" en lugar de ser declarado

La inferencia puede realizarse de acuerdo al tipo de:

- Un operador predefinido — `fun f1(n,m)=(n mod m=0)`
- Un operando — `fun f2(n) = (n*2)`
- Un argumento — `fun f3(n:int) = n*n`
- El tipo del resultado — `fun f4(n):int = (n*n)`

Sistemas de Tipos: nivel de polimorfismo

Un lenguaje se dice **mono-mórfico** si cada entidad se liga a un único tipo

Cada entidad (variable, constante o función) se declara de un tipo específico

En un lenguaje mono-mórfico los chequeos y las ligaduras puede establecerse en forma estática

24

Sistemas de Tipos: nivel de polimorfismo

Un lenguaje se dice **polimórfico** si las entidades pueden estar ligadas a más de un tipo

Las **variables polimórficas** pueden tomas valores de diferentes tipos

Las **operaciones polimórficas** son funciones que aceptan operandos de varios tipos

Los **tipos polimórficos** tienen operaciones polimórficas

25

Sistemas de Tipos: nivel de polimorfismo

Veamos los siguientes ejemplos:

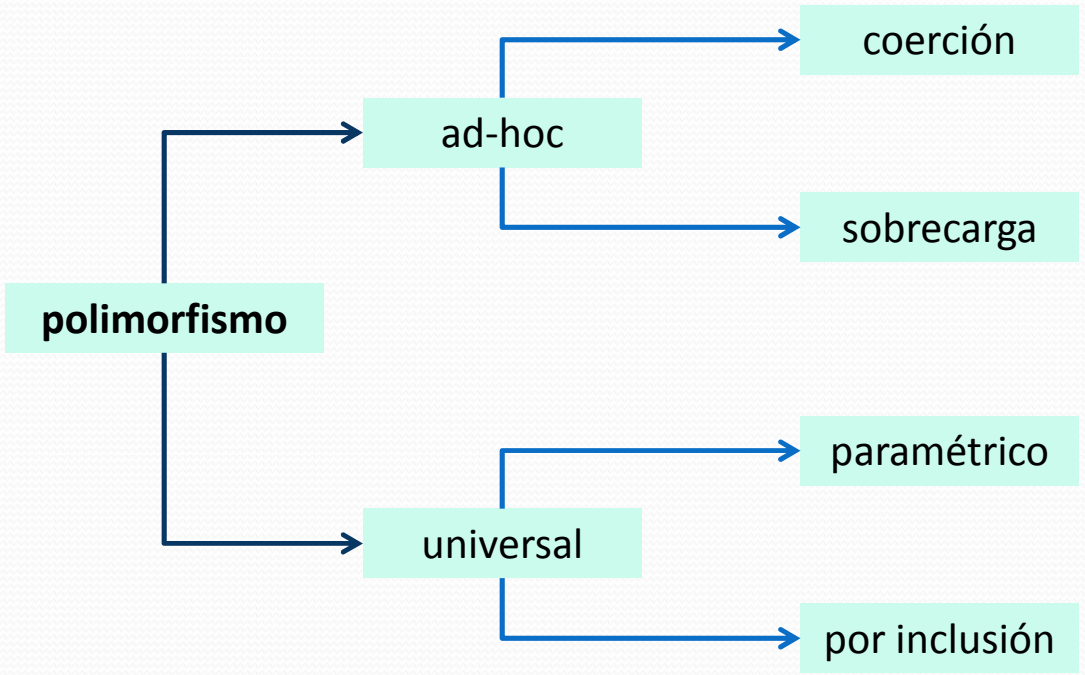
```
a+B
write(e,f(x)+1)
eof(f)
disjuntos(u,v)
fun disjuntos(s1,s2:Conjunto):boolean;
    begin disjuntos:= (s1*s2 = []) end
```

Los lenguajes mono-mórficos son seguros pero inflexibles

- En un lenguaje de programación mono-mórfico la función disjuntos deberá implementarse para cada tipo particular de conjunto.
- Las funciones read y write son “polimórficas”, pero no de forma pura (ya que el compilador infiere el tipo)

26

Sistemas de Tipos: nivel de polimorfismo



27

Sistemas de Tipos: polimorfismo ad-hoc

El **polimorfismo ad-hoc** permite que una función se aplique a distintos tipos con un comportamientos sustancialmente diferente en cada caso

El término **sobrecarga** se utiliza para referirse a conjuntos de abstracciones diferentes que están ligadas al mismo símbolo o identificados

La **coerción** permite que un operador que espera un operando de un determinado tipo T puede aplicarse de manera seguro sobre un operando de un tipo diferente al esperado

28

Sistemas de Tipos: sobrecarga

Cuando hay **sobrecarga** un mismo símbolo denota dos o mas entidades diferentes, pero que en cada llamada particular queda ligado a una función específica

Criterios para resolver la sobrecarga:

- **independientes del contexto** (los datos de entrada de la función son de tipos diferentes)
- **dependientes del contexto** (los datos de entrada son del mismo tipo, no así los de salida)

La incorporación de sobrecarga al lenguaje puede traer inconvenientes si se combina con otras características como el tipo unión, la utilización de parámetros por omisión y las expresiones mixtas

29

Sistemas de Tipos: sobrecarga en los LP

```
int fun f(int x);
float fun f(int x);

int A,B;
A = B + fn f(2);
```

Si la operación + admite operandos mixtos no puede establecerse la ligadura.

C++, Java y C# admiten expresiones mixtas pero la sobrecarga es siempre libre del contexto.

Ada soporta sobrecarga dependiente del contexto pero no con expresiones mixtas

30

Sistemas de Tipos: polimorfismo universal

El **polimorfismo universal** permite que una única operación uniformemente sobre un conjunto de tipos relacionados

Si la uniformidad de la estructura de tipos está dada a través de parámetros, hablamos de **polimorfismo paramétrico**

El **polimorfismo por inclusión** es otra forma de polimorfismo universal que permite modelar subtipos y herencia

31

Sistemas de Tipos: polimorfismo paramétrico

Un **tipo parametrizado** es un tipo que tiene otros tipos como parámetros

En un lenguaje hipotético podrían definirse tipos parametrizados:

par(T) = T x T

lista(T) = T*

En **ML** el concepto de polimorfismo está soportado en el concepto de **politipo**

Tipo que contiene una o más variables de tipo

```
datatype T list = nil | cons of (T * T list)
longitud(L: T list) =
    case L of
      nil => 0
    | cons(h,t) => 1 + longitud(t)
```

32

Sistemas de Tipos: polimorfismo paramétrico

Los lenguajes funcionales, en general proveen este tipo de polimorfismo.

En este contexto muchas veces aparece vinculado a algún mecanismo de inferencia de tipos.

Así se tienen los siguientes casos:

- si el operador de una expresión es mono-mórfico la inferencia de tipos infiere un **monotipo** para la función.
- si el operador de polimórfico la inferencia de tipos producirá un **politipo**

Sistemas de Tipos: polimorfismo por inclusión

El **polimorfismo por inclusión** es un mecanismo para soportar la especialización de un tipo T en otro tipo T'

Existen formas limitadas de polimorfismo por inclusión:

- subtipos
- herencia (mecanismo sumamente poderoso)

34

Sistemas de Tipos: subtipos

Si un tipo se define como un conjunto de valores y un conjunto de operaciones. Un **subtipo** **T'** de un tipo **T** puede definirse como un subconjunto de los valores de **T** y el mismo conjunto de operaciones

T' es un **subtipo** de **T**, si **T'**$\subseteq$**T** de modo que cualquier valor de **T'** puede ser usado de manera segura en cualquier expresión en que se espera un valor de tipo **T**

Ejemplo:

- Los subrangos de Pascal
- los subtipos de Ada

```
type Vector is array(integer range <>) of integer
subtype Vect is Vector(0..99)
```

35

Sistemas de Tipos: herencia

El mecanismo de **herencia** permite definir una nueva clase **derivada** a partir de una clase **base** ya existente

Cuando entre la clase derivada y la clase base existe una relación de tipo "*is a*", toda instancia de la clase derivada es también instancia de la clase base

La clase derivada **hereda** los atributos y el comportamiento de la clase base, puede agregar atributos y comportamiento y redefinir comportamientos de manera compatible

Decimos que la clase derivada es un **subtipo** de la clase base

36

Sistemas de Tipos: herencia

En el uso de la relación de herencia se puede dar el caso en el que la clase derivada oculta atributos o comportamiento o redefine operaciones en forma no compatible. En estos casos la relación no es tipo "*is a*" y por lo tanto no estamos en un contexto en el que podamos hablar de polimorfimo por inclusión

Una **variable polimorfica** puede estar asociada a objetos de diferentes clases dentro de la jerarquía de clases.

Si cada objeto va a responder a un mensaje de acuerdo a la clase que pertenece se requiere de vinculación dinámica de código

37

Sistemas de Tipos: herencia y dureza en el tipado

Restricciones para el correcto manejo de tipos:

1. Una variable declara de tipo B no puede vincularse a un objeto instancia de una superclase de B.
2. No pueden redefinirse métodos o se debe conservar la signatura

```
Class Base {
        method p(){ …… } …..
    }
    Class Derivada: public Base {
        method r(){ …… } …..
        method p(){ …… } …..
    }
```

```
Base B
Derivada D

B:= D
```

B.p — está llamada se comporta de acuerdo a la clase Derivada

D:=B
D.r — En esta llamada el compilador no detecta error y en ejecución no hay ningún método r en la clase Base

38

Sistemas de Tipos: herencia y dureza en el tipado

En un lenguaje orientado a objetos con un sistema fuertemente tipado el polimorfismo de las variables se ve limitado.

El costo es la **integridad conceptual** del lenguaje que ve comprometida al imponer fuertes restricciones sobre la herencia

Para evitar las violaciones de tipos se debe poder **sustituir** a un objeto de una clase derivada por un objeto de la clase base en cualquier contexto. Si se puede garantizar la sustitución de objetos, un tipo derivado puede verse como un subtipo del tipo base.

Imponiendo ciertas restricciones en el uso de la herencia se puede garantizar la **sustitución**.

Para los lenguajes con tipado estático, el propiedad de fuertemente tipado esta asociada a un grado de dureza alto en las características vinculada a polimorfismo.

39

Sistemas de Tipos: herencia y dureza en el tipado

Restricciones:

- Extensión del tipo: La clase derivada solo puede extender la funcionalidad de la clase base. Pero no puede modificar o esconder las operaciones provistas por la clase base. Este mecanismo fue adoptado por Ada 95
- Sobre-escritura de la operaciones: Permite que la clase derivada redefina una operación heredada. En este caso para garantizar la sustitución de objetos debemos:
  1. Los parámetros de entrada deben ser de **super-tipos** de los tipos de los parámetros originales de la operación
  2. Los parámetros de salida deben ser **sub-tipos** de los tipos de los parámetros originales de la operación

40

Sistemas de Tipos: herencia y dureza del tipado

C++ y **Java** soportan una versión limitada de la sobre-escritura de operaciones; ya que ambos requieren que la signatura de la operación redefinida coincida con la signatura de la operación dada en la clase base

Eiffel implementa sobre-escritura de las operaciones, tal como lo presentamos

41