Apuntes-LTP.pdf

sadentsiebla

Lenguajes, Tecnologías y Paradigmas de la Programación

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

- Tipos y sistemas de tipos:
  Int, Char, String...
  Ayudan a detectar errores.
  Puede haber lenguajes tipificados o no tipificados.
  Lenguajes tipificados:
  - Implícitos:
    fac 0 = 1
    fac x = x * fac (x-1)
  - Explícitos:
    int x;
    Char c;
    String s;

  Lenguajes no tipificados:
    objeto(llave)
    objeto(pelota)
    cosa(X) <- objeto (X)

  Constructores de Tipo:
    -> Para definir funciones
    x para definir pares
    [] para definir listas

- Polimorfismo
  La suma puede aplicarse a tipos diferentes (enteros, reales...).
  Tipos de polimorfismo:
  - Universal
    - Paramétrico
    - Inclusión
  - Ad-Hoc
    - Coerción
    - Sobrecarga

  Ad-hoc (o aparente): trabaja sobre un número **finito** de tipos no relacionados.

  Universal (o verdadero): trabaja sobre un número potencialmente **infinito** de tipos con cierta estructura común.

  Sobrecarga:
    Existencia de distintas funciones con el mismo nombre.
    - Los operadores aritméticos suelen estar sobrecargados.
    - En Java, la sobrecarga de métodos se realiza cambiando el tipo de los parámetros:
      int myAdd(...){...}
      double myAdd(...){...}

  Coerción:
    Conversión de valores de un tipo a otro.
    Sentencia **Cast**.

- Conversión *implícita* en Java:

```
int num1 = 100;
long num2 = num1;
```

- Conversión explícita en Java:

```
int num1 = 100;
short num2 = (short) num1;
```

Genericidad:

La definición de una función o la declaración de una clase presenta una estructura común a un número potencialmente infinito de tipos.

- Parametrización:

- Entry<Integer, String> elem1 = new Entry<>(3, “Programming”);

Inclusión (Herencia):

Permite la reutilización de funciones.

- Una clase B heredará una clase A cuando queramos que B tenga estructura y comportamientos de la clase A.
- Podremos añadir nuevos atributos/métodos a B.
- public class MountainBike **extends** Bicycle {...}

Calificadores:

- **Private:** ningún atributo es visible.
- **Protected:** Son visibles en la superclase y en el propio paquete pero no desde otros paquetes.
- **Public:** visibles desde cualquier otra clase.
- **Default:** Son visibles desde cualquier clase que esté en el mismo paquete.

Reflexión:

El programa puede ver su propia estructura y manipularse a sí mismo.

Procedimientos y control de flujo:

- **Paso de parámetros:** cuando se hace una llamada a un método o función hay un cambio de contexto que puede hacerse de distintas formas.
- **Ámbito de las variables:** Ver si una variable es visible en un momento de la ejecución (estático o dinámico).
- **Alcance estático:** tiempo de compilación.
- **Alcance dinámico:** tiempo de ejecución.

Gestión de memoria:

Se refiere a los métodos que se encargan de obtener la máxima utilización de la memoria, organizando los procesos y programas que se ejecutan en el S.O. para optimizar el espacio disponible.

PARADIGMAS DE LA PROGRAMACIÓN:

- Imperativo
- Declarativo
  - Funcional
  - Lógico
- Orientado a objetos
- Concurrente

WUOLAH + BBVA

1/6	BBVA está adherido al Fondo de Garantía de Depósitos de Entidades de Crédito de España. La cantidad máxima garantizada es de 100.000 euros por la totalidad de los depósitos constituidos en BBVA por persona.
Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.	

Te regalamos

15€

1

Abre tu Cuenta Online **sin comisiones ni condiciones**

2

Haz una compra igual o superior a 15€ **con tu nueva tarjeta**

3

BBVA te devuelve un **máximo de 15€**

Paradigma imperativo:

Describe la programación como un conjunto de instrucciones que cambian el estado del programa.

- Establece algoritmos
- Concepto básico -> Estado máquina.
- Instrucciones secuenciales.
- Programa estructurado en bloques y módulos.
- Eficiente (efectos laterales).
- Asignación destructiva (las variables se pueden sobrescribir).

Paradigma declarativo:

Describe las propiedades de la solución buscada.

PROGRAMA = LÓGICA + CONTROL.

El programador se centra en aspectos lógicos de la solución.

Comparación imperativo vs declarativo:

Imperativo:

- Programa -> transcripción de un algoritmo
- Instrucciones -> órdenes a la máquina
- Modelo de computación -> máquina de estados
- Variables -> referencias a memoria

Declarativo:

- Programa -> especificación de un problema
- Instrucciones -> fórmulas lógicas
- Modelo de computación ->máquina de inferencias
- Variables -> variables lógicas

Paradigma orientado a objetos:

Encapsular en objetos estado y operaciones.

Objeto: estado + operaciones

Elementos fundamentales:

- abstracción
- encapsulamiento
- modularidad
- jerarquía

Paradigma concurrente:

Ejecución simultánea de múltiples tareas.

Las tareas pueden consistir en un conjunto de procesos.

Problemas asociados a la concurrencia:

- **Corrupción de los datos:** se produce una mezcla incomprensible.
- **Interbloqueos entre procesos:** problema con acceso a recursos.
- **Inanición de un proceso:** acaparación de los recursos por parte de programas exclusivos.
- **Indeterminismo en el orden.**

Definición de hilos:

- Usando herencia (extends)
- Usando interfaces (implements).

Programación paralela:

Aceleración de algoritmos que consumen muchas horas de procesos mediante distribución de los datos y reparto de la carga.

Comparación de programación concurrente vs paralela:

Paralela:

- Objetivo -> eficiencia
- Procesadores -> solo se concibe con varios
- Comunicación -> paso de mensajes y/o memoria compartida

Concurrente:

- Objetivo -> varios procesadores interactuan simultaneamente
- Procesadores -> es compatible con uno
- Comunicación -> paso de mensajes y/o memoria compartida

Paradigma basado en interacción:

Las entradas se monitorizan y las salidas son acciones que se llevan a cabo dinámicamente.

- **Programa interactivo:** es una comunidad de entidades que interactúan siguiendo unas reglas.

Ventaja:

- Simplifica la tarea del programador.

Desventajas:

- Modelo de interacción excesivamente simple.
- Es difícil de extender.
- Es propenso a errores.

TEMA 2: FUNDAMENTOS DE LOS LENGUAJES DE PROGRAMACIÓN:

Descripción formal de un LP:

- **Sintaxis:** secuencia de caracteres que constituye un programa *legal*.
- **Semántica:** qué significa un programa dado.

Sintaxis:

Notación BNF:

- Con **<w>** se nombra un grupo de expresiones definido por alguna regla de construcción de expresiones.
- El símbolo **|** significa or.
- Los corchetes **[** y **]** se sitúan alrededor de los items opcionales.
- Las llaves **{}** (o el asterisco *****) sirven para indicar una secuencia de 0 o más ítems.
- El símbolo **+** sirve para indicar una secuencia de 1 o más ítems.

Semántica Dinámica:

- Operacional
- Axiomática

- Declarativa:

- Denotacional
- Algebraica
- Teoría de modelos
- Punto Fijo

Semántica Operacional:

Consiste en definir una máquina y expresar el significado de cada construcción del lenguaje en términos de acciones a realizar por dicha máquina.
Representamos el estado de la máquina que ejecuta el programa como una función:

s : X -> D

que asigna valores en un dominio D a las variables del programa.

Notación:

Puesto que tenemos un conjunto finito de variables, podemos representar el estado como un conjunto de pares **variable-valor**.

La configuración de la máquina es un par:

<i, s>

que registra el estado actual (s) junto a la instrucción a evaluar (i).

Para formalizar la ejecución de un programa hemos de establecer una relación de transición (**->**) entre configuraciones.

La relación se define mediante reglas de transición:

premisa

<i, s> -> <i', s'>

El lenguaje SIMP:

Gramática (estilo BNF):

- Expresiones aritméticas:
  - a::= C | V | a1 + a2 | a1 - a2 | a1 * a2
  - C y V denotan las constantes numéricas y las variables.
- Expresiones booleanas:
  - b ::= true | false | a1 = a2 | a1 <= a2 | !b | b1 V b2
- Instrucciones:
  - i ::= skip | V:=a | i1;i2 | if b then i1 else i2 | while b do i
  - donde skip denota la instrucción vacía

- Escribimos <exp, s> -> n para indicar que la expresión exp se evalúa a n en el estado s.
- Usamos esta relación de evaluación para evaluar expresiones aritméticas y booleanas.

Evaluación de expresiones aritméticas:

- Constantes numéricas:
  - <n, s> -> n
- Variables:
  - <x, s> -> s(x)

El estado s es una función de variables en valores. s(x) no es más que el valor de la variable x en el estado de la máquina s.

- Adición:

$$\frac{\langle a1, s\rangle \to n1 \quad \langle a2, s\rangle \to n2}{\langle a1 + a2, s\rangle \to n}$$

si n es la suma de n1 y n2.

- Resta y producto: similar.
- Valores booleanos:

<false, s> -> false <true, s> -> true

- Igualdad:

$$\frac{\langle a1, s\rangle \to n1 \quad \langle a2, s\rangle \to n2}{\langle a1 = a2, s\rangle \to true}$$ si n1 y n2 son iguales

$$\frac{\langle a1, s\rangle \to n1 \quad \langle a2, s\rangle \to n2}{\langle a1 = a2, s\rangle \to false}$$ si n1 y n2 son distintos

- Menor o igual:

$$\frac{\langle a1, s\rangle \to n1 \quad \langle a2, s\rangle \to n2}{\langle a1 <= a2, s\rangle \to true}$$ si n1 es menor o igual que n2

$$\frac{\langle a1, s\rangle \to n1 \quad \langle a2, s\rangle \to n2}{\langle a1 <= a2, s\rangle \to false}$$ si n1 es mayor que n2

- Negación:

$$\frac{\langle b, s\rangle \to true}{\langle !b, s\rangle \to false} \qquad \frac{\langle b, s\rangle \to false}{\langle !b, s\rangle \to true}$$

- Disyunción (hecha por mí xoxo):

$$\frac{\langle a, s\rangle \to n1 \quad \langle b, s\rangle \to n2}{\langle a \vee b, s\rangle \to true}$$ n1 y n2 son disjuntos

$$\frac{\langle a, s\rangle \to n1 \quad \langle b, s\rangle \to n2}{}$$

_____ n1 y n2 no son disjuntos

<a ∨ b, s> -> false

- Secuencia:

<i1, s> -> <i1', s>

_____ _____

<skip; i, s> -> <i, s> <i1; i2, s> -> <i1'; i2, s'>

- Asignación:

<a, s> -> n

_____

<x:=a, s> -> <skip, s[x→n]>

Donde el nuevo estado s[x→n] se define eliminando de s el posible vínculo que exista para x añadiendo el vínculo x→n.

- Condicional:

<b, s> -> true

_____

<if b then i1 else i2, s> -> <i1, s>

<b, s> -> false

_____

<if b then i1 else i2, s> -> <i2, s>

- Bucle while:

<b, s> -> false

_____

<while b do i, s> -> <skip, s>

<b, s> -> true

_____

<while b do i, s> -> <i, while b do i, s>

Ejercicio: Definir la semántica de un while con una única regla utilizando la instrucción condicional:

<b, s> -> true

_____ **MAL**

<while b do i, s> -> <i, s>

Paso Pequeño (Small-Step) - Semántica Operacional

La ejecución de un programa se puede seguir instrucción a instrucción.

Al ejecutar un programa a partir del estado inicial (s1 = {}), se obtiene una secuencia de configuraciones denominada traza.
Existen dos situaciones:

- Pn es la instrucción vacía (skip) para algún n >= 1. La ejecución del programa termina con un estado final sF = sn.
- Pn nunca llega a ser la instrucción vacía para ningún n, ergo la ejecución del programa no termina.

Paso Grande (Big-Step) - Semántica Operacional

Se especifica la ejecución de un programa P como una transición directa desde la configuración inicial <P, s1> al estado final sF.

- Instrucción vacía:

$$\frac{}{\langle skip, s\rangle \Downarrow s}$$

- Secuencia:

$$\frac{\langle i1, s\rangle \Downarrow s1 \qquad \langle i2, s\rangle \Downarrow s'}{\langle i1; i2, s\rangle \Downarrow s'}$$

- Asignación:

$$\frac{\langle a, s\rangle \to n}{\langle x:=a, s\rangle \Downarrow s[x \to]}$$

- Condicional:

$$\frac{\langle b, s\rangle \to true \qquad \langle i1, s\rangle \Downarrow s'}{\langle if\ b\ then\ i1\ else\ i2, s\rangle \Downarrow s'}$$

$$\frac{\langle b, s\rangle \to false \qquad \langle i2, s\rangle \Downarrow s'}{\langle if\ b\ then\ i1\ else\ i2, s\rangle \Downarrow s'}$$

- Bucle while:

$$\frac{\langle b, s\rangle \to false}{\langle while\ b\ do\ i, s\rangle \Downarrow s}$$

$$\frac{\langle b, s\rangle \to true \qquad \langle i, s\rangle \Downarrow s' \qquad \langle while\ b\ do\ i, s'\rangle \Downarrow s''}{\langle while\ b\ do\ i, s\rangle \Downarrow s''}$$

Semántica de un programa:

La semántica de un programa S(P) se define mediante las descripciones operacionales siguientes:

- S*small*(P) es la traza finita (única).
- S*big*(P) es el estado final sF.

Ambas están relacionadas (sF). Pero S*big* tiene un nivel de abstracción mayor que S*small*.

Ejercicio: Calcular la semántica de:

P = (x:=4; while x>3 do x:=-1)

<a, s> -> 4

<while a>3 do a-1, s> -> <a, s>

Semántica axiomática:

Una terna de Hoare (Hoare triple) {P} S {Q} representa la corrección de un programa S respecto a :

- P: una precondición que restringe los estados de entrada a S
- Q: una postcondición que representa los estados de salida deseados.

Corrección de un programa:

Siempre que un estado s satisface P, el estado final s' resultante de ejecutar S satisfará Q.

Existe un *transformador de predicados* que asocia a cada tipo de instrucción (i) y postcondición (Q) una **precondición más débil** = pmd(i, Q).

Dicha pmd ha de cumplir el estado anterior a la ejecución de i para que después de ejecutarse, se garantice Q.

Pasos para la corrección de un programa:

1. Calcular P' = pmd(S, Q)
2. Comprobar que P -> P'

Transformador de predicados pmd:

- Asignación:

  pmd(x:=a, Q) = Q[x -> a]

Aquí x -> a es una sustitución que reemplaza una variable x en una expresión por otra expresión a. Así, Q[x -> a] es el resultado de aplicar esa sustitución a la expresión lógica Q..

- Condicional:

  pmd(if b then i1 else i2, Q) =

  $(b \wedge pmd(i_1,Q)) \vee (\neg b \wedge pmd(i_2,Q))$

- Secuencia:

  pmd(i1; i2, Q) = pmd(i1, pmd(i2, Q))

Ejemplo de cálculo con pmd:

{P} = {x = 0 ∧ y = 1 ∧ z = 2}
{P1} t:=x
{P2} x:=y
{P3} y:=t
{Q} = {x = 1 ∧ y = 0}

1. Cálculo (de abajo a arriba) de P':
   P3 = pmd(y:=t, Q) = Q[y->t] = (x=1 ∧ t=0)
   P2 = pmd(x:=y, P3) = P3[x->y] = (y = 1 ∧ t = 0)
   P1 = pmd(t:=x, P2) = P2[t -> x] = (y =1 ∧ x = 0)
2. Como P1 = pmd(S, Q), comprobamos P->P1.
   (x=0 ∧ y = 1 ∧ z = 2) -> (y = 1 ∧ x = 0)
   Que es cierto.

Propiedades Semánticas:

Equivalencia semántica:

Dos programas P y P' son equivalentes respecto a una descripción semántica si y sólo si:

$$S(P) = S(P')$$

Esto se denota escribiendo $P \equiv_S P'$.

Implementación de los lenguajes de programación:

- Lenguajes compilados:
  Programa Fuente -> Compilador -> Programa Objeto
  Entrada -> Programa Objeto -> Salida
- Lenguajes interpretados:
  Programa Fuente
  -> Intérprete -> Salida
  Entrada

  Los buenos entornos incluyen tanto intérprete como compilador.

FIN :D