# Algoritmos y Estructuras de Datos I

Primer cuatrimestre de 2013

Departamento de Computación - FCEyN - UBA

Especificación - clase 4

Tipos compuestos



## Tipos compuestos

- cada valor de un tipo básico representa un elemento atómico e indivisible
  - Int
  - Float
  - Bool
  - Char
- un valor de un tipo compuesto contiene información que puede ser dividida en componentes de otros tipos
- ya vimos dos ejemplos de tipos compuestos:
  - secuencias
  - tuplas
- para definir un tipo compuesto, tenemos que darle
  - un nombre
  - uno o más observadores



## Observadores



## Observadores

- funciones que se aplican a valores del tipo compuesto y devuelven el valor de sus componentes (pueden tener más parámetros)
- **Ejemplo:** Tipo Punto
  1. sus elementos representan puntos en el plano
  2. componentes: coordenadas ($x$ e $y$)
  3. un observador para obtener cada una
- **Definición:**

```
tipo Punto {
    observador X(p : Punto) : Float;
    observador Y(p : Punto) : Float
}
```

## Observadores

- los observadores definen el tipo compuesto
  - si dos términos del tipo dan el mismo resultado para todos los observadores, entonces se los considera iguales
  - esta condición define la igualdad == para tipos compuestos (el operador == existe en todos los tipos)
- los observadores pueden tener precondiciones
  - si vale la precondición, no pueden devolver un valor indefinido
- los observadores no tienen poscondiciones propias
- si hay condiciones generales que deben cumplir los elementos del tipo, se las presenta como invariantes de tipo



## Más sobre el tipo Punto

- **Ejemplo:** Especificación de una función que recibe dos números reales y construye un punto con esas coordenadas:

```
problema nuevoPunto(a, b : Float) = res : Punto {
    asegura X(res) == a;
    asegura Y(res) == b;
}
```

- Cuando el resultado es de un tipo compuesto, usamos los observadores para describir el resultado
- **Ejemplo:** Función auxiliar para calcular la distancia entre dos puntos

  aux $dist(p, q : \text{Punto}) : \text{Float} = \left((X(p) - X(q))^2 + (Y(p) - Y(q))^2\right)^{1/2}$



## Tipo Círculo

Sus elementos representan círculos en el plano, formados por un **Punto** que es su centro, y un real positivo que es su radio.



## Tipo Círculo

- sus componentes son de tipos distintos

```
tipo Círculo {
    observador Centro(c : Círculo) : Punto;
    observador Radio(c : Círculo) : Float;
    invariante Radio(c) > 0;
}
```

- el invariante del tipo especifica las condiciones que deben cumplir entre sí los valores obtenidos por los observadores para cualquier elemento del tipo
- es la garantía de que los elementos estarán bien construidos

## Invariantes de tipo

- los problemas que reciben valores del tipo compuesto como argumentos pueden suponer que se cumplen los invariantes

tipo T {
  observador ...;
  ⋮
  invariante $R(x)$; }

problema probA(..., $x$ : T, ...) = ... {
  requiere ... ∧ $\underbrace{R(x)}_{\text{no hace falta}}$ ∧ ...;
  asegura ...;
}



## Tipo Círculo

- **Ejemplo:** Especificar el problema de construir un círculo a partir de su centro y su radio:

problema nuevoCírculo($c$ : Punto, $r$ : Float) = $res$ : Círculo {
  requiere $r > 0$;
  asegura ($Centro(res) == c \wedge Radio(res) == r$);
}

- **Ejemplo:** Especificar el problema de construir un círculo a partir de su centro y un punto sobre la circunferencia:

problema nuevoCírculoPuntos($c, x$ : Punto) = $res$ : Círculo {
  requiere $dist(c, x) > 0$;
  asegura ($Centro(res) == c \wedge Radio(res) == dist(c, x)$);
}



## Tipos genéricos

- en los ejemplos vistos, cada componente es de un tipo determinado de antemano
- también hay tipos compuestos que representan una estructura
  - su contenido van a ser valores no siempre del mismo tipo
  - comportamiento uniforme
- se llaman tipos genéricos o tipos paramétricos
  - definen una familia de tipos, y cada elemento de la familia es una instancia del tipo genérico
- los problemas que usen un tipo genérico definen una familia de problemas



## Tipo Matriz$\langle T \rangle$

- tipo genérico de las matrices cuyos elementos pertenecen a un tipo cualquiera $T$

tipo Matriz$\langle T \rangle$ {
  observador $filas(m$ : Matriz$\langle T \rangle) : Int$;
  observador $columnas(m$ : Matriz$\langle T \rangle) : Int$;
  observador $val(m$ : Matriz$\langle T \rangle, f, c : Int) : T$ {
    requiere $0 \leq f < filas(m)$;
    requiere $0 \leq c < columnas(m)$;
  } } precondición del observador
  invariante $filas(m) > 0$;
  invariante $columnas(m) > 0$;
}

- ejemplo: una instancia del tipo genérico Matriz$\langle T \rangle$ es Matriz$\langle$Char$\rangle$

## Operaciones genéricas con matrices

- expresión que cuenta la cantidad de elementos de una matriz

  aux $elementos(m : \text{Matriz}\langle T\rangle) : \text{Int} = filas(m) * columnas(m)$;

- especificación del problema de cambiar el valor de una posición de una matriz

  problema cambiar($m : \text{Matriz}\langle T\rangle, f, c : \text{Int}, v : T$) {
  requiere $0 \leq f < filas(m)$;
  requiere $0 \leq c < columnas(m)$;
  modifica $m$;
  asegura $filas(m) == filas(pre(m))$;
  asegura $columnas(m) == columnas(pre(m))$;
  asegura $val(m, f, c) == v$;
  asegura $(\forall i \leftarrow [0..filas(m)))$
  $(\forall j \leftarrow [0..columnas(m)),\ \neg(i == f \wedge j == c))$
  $val(m, i, j) == val(pre(m), i, j)))$;
  }



## Operaciones sobre matrices para tipos instanciados

- expresión que suma los elementos de una matriz de enteros

  aux $suma(m : \text{Matriz}\langle\text{Int}\rangle) : \text{Int} =$
  $\sum[val(m, i, j) \mid i \leftarrow [0..filas(m)), j \leftarrow [0..columnas(m))]$;

- especificación del problema de construir la matriz identidad de $n \times n$:

  problema matId($n : \text{Int}$) = $ident : \text{Matriz}\langle\text{Int}\rangle$ {
  requiere $n > 0$;
  asegura $filas(ident) == columnas(ident) == n$;
  asegura $(\forall i \leftarrow [0..n))\ val(ident, i, i) == 1$;
  asegura $(\forall i \leftarrow [0..n))(\forall j \leftarrow [0..n), i \neq j)\ val(ident, i, j) == 0$;
  }

  - ¿importa el orden?
  - sí; si estuviera al revés, se podría indefinir $val(ident, i, i)$
  - en este caso, se indefiniría la poscondición, valiendo la precondición



## Secuencias



## El tipo Secuencia$\langle T\rangle$

- ya lo usamos con su nombre alternativo: $[T]$
- presentamos también sus observadores: longitud e indexación

  tipo Secuencia$\langle T\rangle$ {
  observador $long(s : \text{Secuencia}\langle T\rangle) : \text{Int}$;
  observador $índice(s : \text{Secuencia}\langle T\rangle, i : \text{Int}) : T$ {
  requiere $0 \leq i < long(s)$; $\longrightarrow$ precondición de observador
  }
  }

- notaciones alternativas
  - $|s|$ para la longitud
  - $s_i$ o $s[i]$ para la indexación

## Operaciones genéricas con secuencias

- podemos definir
  - aux $ssc(a, b : [T]) : Bool =$
    $(\exists i \leftarrow [0..|b| - |a|))\ a == b[i..(i + |a|))$;
  - aux $cab(a : [T]) : T = a[0]$;
  - aux $cola(a : [T]) : [T] = a[1..|a|)$;
  - aux $en(t : T, a : [T]) : \text{Bool} = [x \mid x \leftarrow a, x == t] \neq [\,]$;
  - aux $sub(a : [T], d, h : \text{Int}) : [T] =$
    $[a[i] \mid i \leftarrow [d..h], (0 \geq d \ \wedge\ h < |a|)]$;
  - aux $todos(sec : [\text{Bool}]) : \text{Bool} = \text{false} \notin sec$;
  - aux $alguno(sec : [\text{Bool}]) : \text{Bool} = \text{true} \in sec$;
- aprovechamos la notación de listas por comprensión
- el selector es parte de la notación de listas por comprensión (estructura especial de nuestro lenguaje de especificación)



## Operaciones sobre secuencias para tipos instanciados

- para representar cadenas de caracteres (strings) usamos el tipo Secuencia⟨Char⟩ (o [Char])
- queremos ver si alguna palabra de una lista aparece en un libro

```
problema hayAlguna(palabras : [[Char]], libro : [Char]) = res : Bool {
    requiere NoVacías : (∀p ← palabras) |p| > 0;
    requiere SinEspacios : ¬(∃p ← palabras) ' ' ∈ p;
    asegura res == (∃p ← palabras) ssc(p, libro);
}
```



## Tuplas

- secuencias de tamaño fijo, en las que cada elemento puede pertenecer a un tipo distinto (predeterminado de antemano)

```
tipo Par⟨A, B⟩{
    observador prm(p : Par⟨A, B⟩) : A;
    observador sgd(p : Par⟨A, B⟩) : B;
}

tipo Terna⟨A, B, C⟩{
    observador prm3(t : Terna⟨A, B, C⟩) : A;
    observador sgd3(t : Terna⟨A, B, C⟩) : B;
    observador trc3(t : Terna⟨A, B, C⟩) : C;
}
```

- **Notación:**
  1. Par⟨A, B⟩ también se puede escribir $(A, B)$
  2. Terna⟨A, B, C⟩ también se puede escribir $(A, B, C)$



## ifThenElse⟨T⟩

- Función que elige entre dos elementos del mismo tipo, según una condición (guarda)
- si la guarda es verdadera, elige el primero; si no, elige el segundo
- **Ejemplo:** expresión que devuelve el máximo entre dos elementos:

  aux máx$(a, b : \text{Int}) : \text{Int} = \text{ifThenElse}\langle\text{Int}\rangle(a > b, a, b)$

- cuando el tipo de los argumentos se deducen del contexto, se puede escribir directamente

  aux máx$(a, b : \text{Int}) : \text{Int} = \text{ifThenElse}(a > b, a, b)$ o bien
  aux máx$(a, b : \text{Int}) : \text{Int} = \text{if } a > b \text{ then } a \text{ else } b$

- **Ejemplo:** expresión que dado $x$ devuelve $1/x$ si $x \neq 0$ y 0 sino

  aux unoSobre$(x : \text{Float}) : \text{Float} =$ $\underbrace{\text{if } x \neq 0 \text{ then } 1/x \text{ else } 0}_{\text{no se indefine cuando } x = 0}$

## Más aplicaciones de IfThenElse

- agregar un elemento como primer elemento de una lista

  aux cons($x : T, a : [T]) : [T] =$
  $\quad$ [if $i == -1$ then $x$ else $a[i]$ | $i \leftarrow [-1..|a|)$];

- concatenar dos listas

  aux conc($a, b : [T]) : [T] =$
  $\quad$ [if $i < |a|$ then $a[i]$ else $b[i - |a|]$ | $i \leftarrow [0..|a| + |b|)$];

- cambiar el valor de una posición

  aux cambiar($a : [T], i : \text{Int}, v : T) : [T] =$
  $\quad$ [if $i \neq j$ then $a[i]$ else $v$ | $j \leftarrow [0..|a|)$];