Data Structures.
Grado en Informática, Ingeniería del Software y Computadores
ETSI Informática
Universidad de Málaga

El Tipo Abstracto de Datos Bolsa

@ Pablo López
Dpto. Lenguajes y Ciencias de la Computación
Universidad de Málaga

Metodología de Trabajo

1. Especificar informalmente el TAD
2. Especificar formalmente el TAD
    1. Interfaz
    2. Especificación algebraica
3. Implementar el TAD
4. Analizar la eficiencia
5. Usar el TAD para resolver problemas

Especificación Informal

- Una bolsa es similar a un conjunto, pero los elementos pueden aparecer varias veces:

```
{ 'a', 'b', 'c', 'f', 'a', 'a', 't', 'c', 'a', 'c' }
```

- Como en cualquier colección, podremos:
  - **Insertar** un dato
  - **Borrar** un dato
  - **Comprobar** si la colección está **vacía**
  - **Consultar** cuántas veces aparece un dato

Especificación formal: interfaz

El TAD `Bag a` tiene las siguientes operaciones:

```
-- constructores
empty :: Bag a
insert :: Ord a => a -> Bag a -> Bag a

-- selectores
isEmpty :: Bag a -> Bool
occurrences :: Ord a => a -> Bag a -> Int

--transformadores
delete :: Ord a => a -> Bag a -> Bag a
```

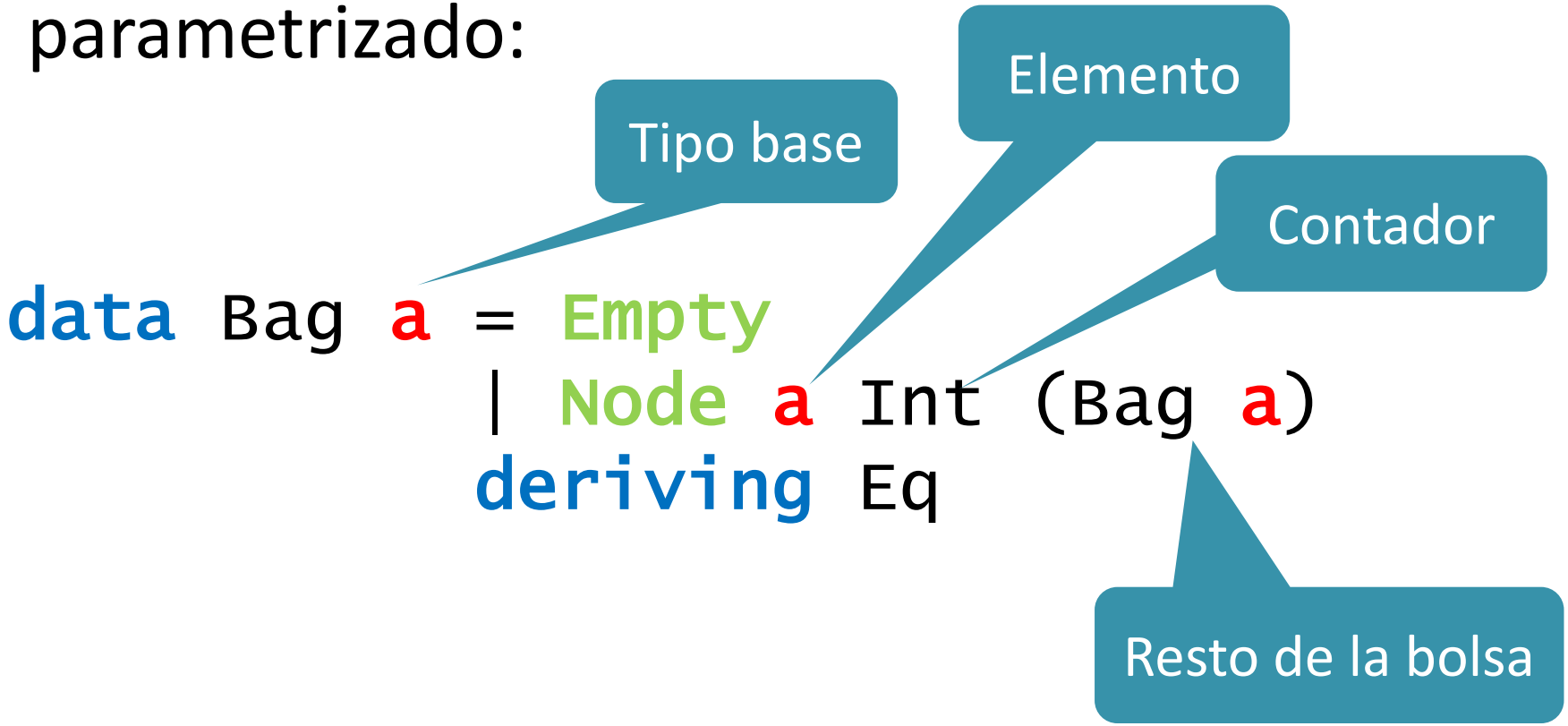
Especificación formal: axiomas

- Basta especificar el resultado que deben devolver los **selectores** y **transformadores** cuando se aplican a una bolsa obtenida con los **constructores** `empty` e `insert`

- Las especificaciones de `occurrences` y `delete` deben distinguir dos casos para el constructor `insert`, según el dato esté o no presente en la bolsa

Implementación del TAD Bolsa (I)

- Utilizaremos el siguiente tipo algebraico parametrizado:



Implementación de la Bolsa (II)

- La bolsa:

```
{ 'a', 'b', 'c', 'a', 'a', 't', 'c', 'a', 'c' }
```

se presenta en Haskell por:

```
Node 'a' 4 (Node 'b' 1 (Node 'c' 3 (Node 't' 1 Empty)))
```

Contadores positivos

Ordenado por elemento, sin repetidos

Implementación de la Bolsa: insert

```
insert 'a' Empty =
    Node 'a' 1 Empty

insert 'a' (Node 'a' 5 (Node 'c' 3 Empty)) =
    Node 'a' 6 (Node 'c' 3 Empty)

insert 'b' (Node 'a' 5 (Node 'c' 3 Empty)) =
    Node 'a' 5 (Node 'b' 1 (Node  'c' 3 Empty))

insert 'w' (Node 'a' 5 (Node 'c' 3 Empty)) =
    Node 'a' 5 (Node 'c' 3 (Node 'w' 1 Empty))
```

Operaciones auxiliares sobre bolsas

```
b1 = Node ‘a’ 2 (Node ‘b’ 3 Empty)
b2 = Node ‘a’ 5 (Node ‘c’ 1 Empty)

union b1 b2 =
    Node ‘a’ 7 (Node ‘b’ 3 (Node ‘c’ 1 Empty))

intersection b1 b2 =
    Node ‘a’ 2 Empty

difference b1 b2 =
    Node ‘b’ 3 Empty

difference b2 b1
    Node ‘a’ 3 (Node ‘c’ 1 Empty)
```

Eficiencia de la implementación

- Para cada operación, determinar el **número de pasos** que deben realizarse para llevarla a cabo:
  - **O(1)** – número de pasos constante, independiente del tamaño de la bolsa
  - **O(n)** – el número de pasos es proporcional al tamaño de la bolsa

Plegado de Bolsa: tipo

Para manejar bolsas como cliente es necesario utilizar el siguiente plegado:

```
foldBag :: Ord a =>
           (a -> Int -> b -> b) ->
           b ->
           Bag a ->
           b
```

f x ox solRestoBolsa

Caso base

Plegado de Bolsa: implementación

```
foldBag f z Empty = z
foldBag f z (Node x ox s) =
                    f x ox (foldBag f z s)
```

Uso del plegado

- Obtener una lista (sin repeticiones) con los elementos de una bolsa:

```
keys :: Ord a => Bag a -> [a]
keys bag = foldBag (\e oe s -> e:s) [] bag
```

- Ejemplo de uso:

```
keys (list2Bag "abracadabra“)
"abcdr"
```