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

El Tipo Abstracto de Datos Bolsa

@ Pablo López, @ José E. Gallardo, @ Francisco Gutiérrez
Dpto. Lenguajes y Ciencias de la Computación
Universidad de Málaga

Especificación Informal

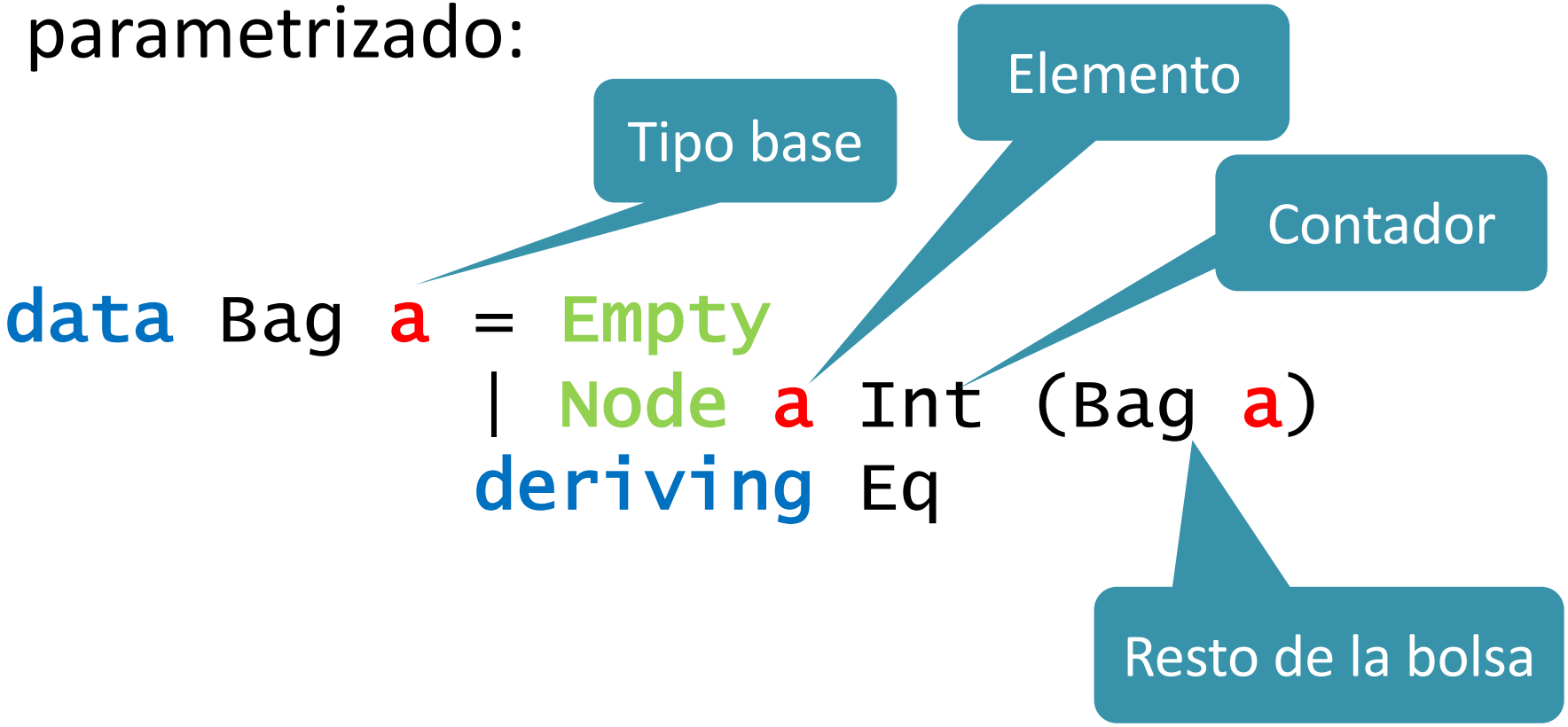
- Una bolsa es similar a un conjunto, pero los elementos pueden aparecer varias veces:

```
{ '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

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

Nunca puede haber un elemento con 0 apariciones

Especificación formal: interfaz

Ejercicio 1:

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
```

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))
```

La interfaz se puede probar con checkBag (ejercicio 3)

Función auxiliar. list2Bag

Ejercicio 2:

Dada una lista, construye con plegado un bag con sus elementos.

```
list2Bag:: Ord a => [a] -> Bag a
list2Bag xs = undefined
```

Ejemplo de uso:

```
list2Bag "abracadabra" -->
Node 'a' 5 (Node 'b' 2 (Node 'c' 1 (Node 'd' 1
(Node 'r' 2 Empty))))
```

Especificación formal: axiomas

- Ejercicio 3: Completar los 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

Plegado de Bolsa

Para manejar bolsas definimos el siguiente plegado:

```
foldBag :: Ord a =>
           (a -> Int -> b -> b) ->
           b ->
           Bag a ->
           b
foldBag f z Empty = z
foldBag f z (Node x ox bag) =
              f x ox (foldBag f z bag)
```

Uso del plegado

Ejercicio 4:

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

```
keys :: Ord a => Bag a -> [a]
keys bag = undefined
```

- Ejemplo de uso:

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

Uso del TAD Bolsa (II)

Determina si un elemento pertenece a la bolsa (con plegado)

```
contains :: Ord a => a -> Bag a -> Bool
contains x bag = undefined
```

- Ejemplo de uso:

```
contains 'a' (list2Bag "Haskell") --> True

contains 'd' (list2Bag "Haskell") --> False
```

Complejidad

- Ejercicio 5:
- Completar la tabla de complejidad de las funciones básicas de la interfaz.

- ¿Hay alguna ventaja en mantener los elementos ordenados en una bolsa? ¿Por qué?