

Tipos Abstractos de Datos

Mauro Jaskelioff

10/05/2022



- ▶ ¿Qué es una cola?
- ▶ Es una estructura a la cual:
 - ▶ Podemos agregar elementos
 - ▶ Podemos obtener el primer elemento
 - ▶ Podemos quitar el primer elemento
 - ▶ Podemos preguntar si está vacía
 - ▶ Existe una relación entre el orden en que se agregan elementos y se sacan (FIFO).
- ▶ Esta descripción es *abstracta* porque refleja el comportamiento y no la implementación.

Tipos Abstractos de Datos

- ▶ La idea de un tipo abstracto de datos es abstraer detalles de implementación.
- ▶ Un usuario es alguien que simplemente usa la abstracción.
- ▶ El implementador provee una implementación que se ajusta al comportamiento esperado.
- ▶ El usuario sólo puede suponer el comportamiento descripto.
- ▶ Podemos ser más precisos sobre el comportamiento de las colas mediante una *especificación*.

Un TAD consiste de:

1. Un nombre de tipo. Ej . *Cola*
2. Operaciones.

```
tad Cola (A : Set) where  
  import Bool  
  vacia    : Cola A  
  poner    : A → Cola A → Cola A  
  primero : Cola A → A  
  sacar    : Cola A → Cola A  
  esVacia  : Cola A → Bool
```

3. Especificación del comportamiento

- ▶ Especificación algebraica
 - ▶ Se describen operaciones y ecuaciones entre operaciones
- ▶ Modelos
 - ▶ Se describen operaciones y cómo se interpretan en un modelo matemático

Especificaciones Algebraicas

► Especificación algebraica para colas

$esVacia\ vacia = True$

$esVacia\ (poner\ x\ q) = False$

$primero\ (poner\ x\ vacia) = x$

$primero\ (poner\ x\ (poner\ y\ q)) = primero\ (poner\ y\ q)$

$sacar\ (poner\ x\ vacia) = vacia$

$sacar\ (poner\ x\ (poner\ y\ q)) = poner\ x\ (sacar\ (poner\ y\ q))$

► No confundir especificación con implementación!

Especificaciones Algebraicas (cont.)

- ▶ Las especificaciones deben ser mediante ecuaciones.
 - ▶ Solo deben aparecer operaciones de TAD y variables libres (se suponen cuantificadas universalmente)
 - ▶ Incorrecto:

$$size\ x = \begin{cases} 0 & \text{si } esVacia\ x \\ 1 + size\ (sacar\ x) & \text{en otro caso.} \end{cases}$$

- ▶ Correcto:

$$size\ x = \text{if } (esVacia\ x) \text{ then } 0 \text{ else } 1 + size\ (sacar\ x)$$

- ▶ 0, 1, son operadores nularios, y + es un operador binario de un TAD de Naturales.
 - ▶ **if then else** es un operador ternario de un TAD de Booleanos.
- ▶ Puede quedar comportamiento sin definir.
 - ▶ ¿Qué dice la especificación sobre *primero vacia*?

- ▶ Como modelo de colas tomamos las secuencias $\langle x_1, x_2, \dots, x_n \rangle$
- ▶ Para cada operación damos una función equivalente sobre modelos:

$$\text{vacía} = \langle \rangle$$

$$\text{poner } x \langle x_1, x_2, \dots, x_n \rangle = \langle x, x_1, x_2, \dots, x_n \rangle$$

$$\text{sacar } \langle x_1, x_2, \dots, x_n \rangle = \langle x_1, x_2, \dots, x_{n-1} \rangle$$

$$\text{primero } \langle x_1, x_2, \dots, x_n \rangle = x_n$$

$$\text{esVacía } \langle x_1, x_2, \dots, x_n \rangle = \text{True} \quad \text{si } n = 0$$

$$\text{esVacía } \langle x_1, x_2, \dots, x_n \rangle = \text{False} \quad \text{en otro caso}$$

Trabajando con modelos

- ▶ A cada cola c , le corresponde un modelo $\llbracket c \rrbracket$.
- ▶ A cada operación op , le corresponde un operación $\llbracket op \rrbracket$ que trabaja sobre el modelo.
- ▶ Una implementación se ajusta al modelo si:

$$\llbracket op \ c \rrbracket = \llbracket op \rrbracket \ \llbracket c \rrbracket$$

- ▶ Por ejemplo, esperamos que

$$\llbracket sacar \ c \rrbracket = \llbracket sacar \rrbracket \ \llbracket c \rrbracket$$

- ▶ Cada TAD admite diferentes implementaciones
- ▶ Ejercicio:
 - a) Implementar en Haskell el TAD de colas usando listas.

```
tad Cola (A : Set) where  
  vacia    : Cola A  
  poner    : A → Cola A → Cola A  
  primero : Cola A → A  
  sacar    : Cola A → Cola A  
  esVacia  : Cola A → Bool
```

- b) ¿Cuál es el trabajo de las operaciones?

Costo de las implementaciones

- ▶ Hay dos posibles implementaciones con listas.
 1. Se agregan elementos al final, se sacan de la cabeza.
 2. Se agregan elementos a la cabeza, se sacan del final.
- ▶ Los dos métodos son lineales en alguna operación.
- ▶ ¿Cómo implementar una cola eficiente?

Otra implementación de Colas

- Implementemos colas usando un par de listas (xs, ys) tal que los elementos en orden sean $xs \mathrel{++} reverse\ ys$
- Invariante de la implementación: Si xs es vacía, entonces ys también (las operaciones deben conservar este invariante. . .)

```
type Cola a = ([a], [a])  
vacía           = ([], [])  
poner x (ys, zs) = validar (ys, x : zs)  
prim (x : xs, ys) = x  
sacar (x : xs, ys) = validar (xs, ys)  
esvacía (xs, ys)  = null xs  
validar (xs, ys)  = if null xs then (reverse ys, [])  
                  else (xs, ys)
```

Costo de esta implementación

La implementación con pares tiene los siguientes costos:

- ▶ $W_{vacía} \in O(1)$
- ▶ $W_{esVacía}(xs, ys) \in O(1)$
- ▶ $W_{poner}(xs, ys) \in O(1)$
- ▶ $W_{primero}(xs, ys) \in O(1)$
- ▶ $W_{sacar}(xs, ys) \in O(|ys|), O(1)_{(amortizado)}$

Especificación de costo

- ▶ Cada TAD admite diferentes implementaciones
- ▶ En cada implementación las operaciones pueden tener diferentes costos (trabajo, profundidad).
- ▶ Dependiendo del *uso* de la estructura, puede convenir una implementación u otra.
- ▶ Por lo tanto es importante tener una *especificación de costo* de cada implementación.
 - ▶ Ejemplo: Para la 2da implementación de colas

$$\begin{array}{lll} W_{vacía} \in O(1) & W_{poner}(xs, ys) \in O(1) & W_{primero}(xs, ys) \in O(1) \\ W_{sacar}(xs, ys) \in O(|ys|), O(1)_{(amortizado)} & & W_{esVacía}(xs, ys) \in O(1) \end{array}$$

TADs en Haskell

- Una forma de implementar un TAD en Haskell es mediante una clase de tipos

```
class Cola t where  
    vacia    :: t a  
    poner   :: a → t a → t a  
    sacar   :: t a → t a  
    primero :: t a → a  
    esVacia :: t a → Bool
```

- Una implementación es una instancia

```
instance Cola [] where  
    vacia      = []  
    poner x xs = x : xs  
    sacar xs   = init xs  
    primero    = last  
    esVacia xs = null xs
```

- ▶ Usamos un TAD con una función polimórfica en el TAD

$$\begin{aligned} \text{ciclar} & \quad :: \text{Cola } t \Rightarrow \text{Int} \rightarrow t \rightarrow a \rightarrow t \rightarrow a \\ \text{ciclar } 0 \text{ cola} &= \text{cola} \\ \text{ciclar } n \text{ cola} &= \text{ciclar } (n - 1) (\text{poner } (\text{primero cola}) \\ & \quad \quad \quad (\text{sacar cola})) \end{aligned}$$

- ▶ Notar que la función *ciclar* funciona para cualquier instancia de Cola. O, lo que es lo mismo, funciona para cualquier implementación del TAD.
- ▶ La función *ciclar* no puede suponer nada acerca de la implementación.

- ▶ **Especificación:** Qué operaciones tiene el TAD y cómo se comportan. Es Única.
- ▶ **Implementación:** Cómo se realizan las operaciones y cuánto cuestan. Puede haber varias implementaciones (con diferentes costos). Todas deben garantizar el comportamiento dado por la especificación.
- ▶ **Uso:** Sólo puede suponer el comportamiento dado por la especificación. Se elige implementación de acuerdo al uso (menor costo para un determinado uso).

Tipos Abstractos de Datos

- ▶ Ocultan detalles de implementación.
- ▶ El comportamiento se describe algebraicamente o proveyendo un modelo.
- ▶ Cada implementación debe tener una especificación de costo.