# Tema 14: El TAD de las pilas

## Informática (2015–16)

José A. Alonso Jiménez

Grupo de Lógica Computacional
Departamento de Ciencias de la Computación e I.A.
Universidad de Sevilla

# Tema 14: El TAD de las pilas

# Tema 14: El TAD de las pilas

# Abstracción y tipos abstractos de datos

- La **abstracción** es un mecanismo para comprender problemas que involucran una gran cantidad de detalles.
- Aspectos de la abstracción:
  - **Destacar** los detalles relevantes.
  - **Ocultar** los detalles irrelevantes.
- Un **tipo abstracto de datos** (TAD) es una colección de *valores* y *operaciones* que se definen mediante una *especificación* que es independiente de cualquier *representación*.
- Un TAD es una abstracción:
  - Se destacan los detalles (normalmente pocos) de la especificación (*el qué*).
  - Se ocultan los detalles (normalmente numerosos) de la implementación (*el cómo*).
- Analogía con las **estructuras algebraicas**.

# Tema 14: El TAD de las pilas

# Descripción informal de las pilas

- Una **pila** es una estructura de datos, caracterizada por ser una secuencia de elementos en la que las operaciones de inserción y extracción se realizan por el mismo extremo.
- La pilas también se llaman estructuras LIFO (del inglés Last In First Out), debido a que el último elemento en entrar será el primero en salir.
- Analogía con las pilas de platos.

# Signatura del TAD de las pilas

- Signatura:

```
vacia    :: Pila a
apila    :: a -> Pila a -> Pila a
cima     :: Pila a -> a
desapila :: Pila a -> Pila a
esVacia  :: Pila a -> Bool
```

- Descripción:
  - `vacia` es la pila vacía.
  - `(apila x p)` es la pila obtenida añadiendo `x` al principio de `p`.
  - `(cima p)` es la cima de la pila `p`.
  - `(desapila p)` es la pila obtenida suprimiendo la cima de `p`.
  - `(esVacia p)` se verifica si `p` es la pila vacía.

# Tema 14: El TAD de las pilas

# Propiedades de las pilas

1. `cima (apila x p) == x`
2. `desapila (apila x p) == p`
3. `esVacia vacia`
4. `not (esVacia (apila x p))`

# Tema 14: El TAD de las pilas

# Las pilas mediante tipos de datos algebraicos

- Cabecera del módulo:

```
module PilaConTipoDeDatoAlgebraico
    (Pila,
     vacia,    -- Pila a
     apila,    -- a -> Pila a -> Pila a
     cima,     -- Pila a -> a
     desapila, -- Pila a -> Pila a
     esVacia   -- Pila a -> Bool
    ) where
```

- Tipo de dato algebraico de las pilas.

```
data Pila a = Vacia | P a (Pila a)
              deriving Eq
```

# Las pilas mediante tipos de datos algebraicos

- Procedimiento de escritura de pilas.

```
instance (Show a) => Show (Pila a) where
    showsPrec p Vacia cad   = showChar '-' cad
    showsPrec p (P x s) cad =
       shows x (showChar '|' (shows s cad))
```

- Ejemplo de pila:
  - Definición

```
p1 :: Pila Int
p1 = apila 1 (apila 2 (apila 3 vacia))
```

  - Sesión

```
ghci> p1
1|2|3|-
```

# Las pilas mediante tipos de datos algebraicos

- `vacia` es la pila vacía. Por ejemplo,

```
ghci> vacia
-
```

```
vacia :: Pila a
vacia = Vacia
```

- `(apila x p)` es la pila obtenida añadiedo `x` encima de la pila `p`. Por ejemplo,

```
apila 4 p1  =>  4|1|2|3|-
```

```
apila :: a -> Pila a -> Pila a
apila x p = P x p
```

# Las pilas mediante tipos de datos algebraicos

- `vacia` es la pila vacía. Por ejemplo,

```
ghci> vacia
-
```

```
vacia :: Pila a
vacia = Vacia
```

- `(apila x p)` es la pila obtenida añadiedo `x` encima de la pila `p`. Por ejemplo,

```
apila 4 p1  =>  4|1|2|3|-
```

```
apila :: a -> Pila a -> Pila a
apila x p = P x p
```

# Las pilas mediante tipos de datos algebraicos

- `vacia` es la pila vacía. Por ejemplo,

```
ghci> vacia
-
```

```
vacia :: Pila a
vacia = Vacia
```

- `(apila x p)` es la pila obtenida añadiedo `x` encima de la pila `p`. Por ejemplo,

```
apila 4 p1  =>  4|1|2|3|-
```

```
apila :: a -> Pila a -> Pila a
apila x p = P x p
```

# Las pilas mediante tipos de datos algebraicos

- `(cima p)` es la cima de la pila `p`. Por ejemplo,

```
cima p1  ==  1
```

```
cima :: Pila a -> a
cima Vacia    = error "cima: pila vacia"
cima (P x _) =  x
```

- `(desapila p)` es la pila obtenida suprimiendo la cima de la pila `p`. Por ejemplo,

```
desapila p1  =>  2|3|-
```

```
desapila :: Pila a -> Pila a
desapila Vacia    = error "desapila: pila vacia"
desapila (P _ p) = p
```

# Las pilas mediante tipos de datos algebraicos

- `(cima p)` es la cima de la pila `p`. Por ejemplo,

```
cima p1  ==  1
```

```
cima :: Pila a -> a
cima Vacia   = error "cima: pila vacia"
cima (P x _) =  x
```

- `(desapila p)` es la pila obtenida suprimiendo la cima de la pila `p`. Por ejemplo,

```
desapila p1  =>  2|3|-
```

```
desapila :: Pila a -> Pila a
desapila Vacia   = error "desapila: pila vacia"
desapila (P _ p) = p
```

# Las pilas mediante tipos de datos algebraicos

- `(cima p)` es la cima de la pila `p`. Por ejemplo,

```
cima p1  ==  1
```

```
cima :: Pila a -> a
cima Vacia   = error "cima: pila vacia"
cima (P x _) =  x
```

- `(desapila p)` es la pila obtenida suprimiendo la cima de la pila `p`. Por ejemplo,

```
desapila p1  =>  2|3|-
```

```
desapila :: Pila a -> Pila a
desapila Vacia   = error "desapila: pila vacia"
desapila (P _ p) = p
```

# Las pilas mediante tipos de datos algebraicos

- `(esVacia p)` se verifica si `p` es la pila vacía. Por ejemplo,

```
esVacia p1     ==  False
esVacia vacia  ==  True
```

```
esVacia :: Pila a -> Bool
esVacia Vacia = True
esVacia _     = False
```

# Las pilas mediante tipos de datos algebraicos

- `(esVacia p)` se verifica si `p` es la pila vacía. Por ejemplo,

```
esVacia p1     ==  False
esVacia vacia  ==  True
```

```
esVacia :: Pila a -> Bool
esVacia Vacia = True
esVacia _     = False
```

# Tema 14: El TAD de las pilas

# Implementación de las pilas mediante listas

- Cabecera del módulo

```
module PilaConListas
    (Pila,
     vacia,      -- Pila a
     apila,      -- a -> Pila a -> Pila a
     cima,       -- Pila a -> a
     desapila,   -- Pila a -> Pila a
     esVacia     -- Pila a -> Bool
    ) where
```

- Tipo de datos de las pilas:

```
newtype Pila a = P [a]
    deriving Eq
```

# Implementación de las pilas mediante listas

- Procedimiento de escritura de pilas.

```
instance (Show a) => Show (Pila a) where
    showsPrec p (P [])     cad = showChar '-' cad
    showsPrec p (P (x:xs)) cad
        = shows x (showChar '|' (shows (P xs) cad))
```

- Ejemplo de pila: `p1` es la pila obtenida anadiéndole los elementos 3, 2 y 1 a la pila vacía. Por ejemplo,

```
ghci> p1
1|2|3|-
```

```
p1 = apila 1 (apila 2 (apila 3 vacia))
```

# Implementación de las pilas mediante listas

- `vacia` es la pila vacía. Por ejemplo,

```
ghci> vacia
-
```

```
vacia   :: Pila a
vacia = P []
```

- `(apila x p)` es la pila obtenida añadiendo `x` encima de la pila `p`. Por ejemplo,

```
apila 4 p1  =>  4|1|2|3|-|
```

```
apila :: a -> Pila a -> Pila a
apila x (P xs) = P (x:xs)
```

# Implementación de las pilas mediante listas

- `vacia` es la pila vacía. Por ejemplo,

```
ghci> vacia
-
```

```
vacia   :: Pila a
vacia = P []
```

- `(apila x p)` es la pila obtenida añadiendo `x` encima de la pila `p`. Por ejemplo,

```
apila 4 p1  =>  4|1|2|3|-|
```

```
apila :: a -> Pila a -> Pila a
apila x (P xs) = P (x:xs)
```

# Implementación de las pilas mediante listas

- `vacia` es la pila vacía. Por ejemplo,

```
ghci> vacia
-
```

```
vacia   :: Pila a
vacia = P []
```

- `(apila x p)` es la pila obtenida añadiendo `x` encima de la pila `p`. Por ejemplo,

```
apila 4 p1  =>  4|1|2|3|-|
```

```
apila :: a -> Pila a -> Pila a
apila x (P xs) = P (x:xs)
```

# Implementación de las pilas mediante listas

- `(cima p)` es la cima de la pila `p`. Por ejemplo,

```
cima p1  ==  1
```

```
cima :: Pila a -> a
cima (P (x:_)) = x
cima (P [])    = error "cima de la pila vacia"
```

- `(desapila p)` es la pila obtenida suprimiendo la cima de la pila `p`. Por ejemplo,

```
desapila p1  =>  2|3|-
```

```
desapila :: Pila a -> Pila a
desapila (P [])     = error "desapila la pila vacia"
desapila (P (_:xs)) = P  xs
```

# Implementación de las pilas mediante listas

- `(cima p)` es la cima de la pila `p`. Por ejemplo,

```
cima p1  ==  1
```

```
cima :: Pila a -> a
cima (P (x:_)) = x
cima (P [])    = error "cima de la pila vacia"
```

- `(desapila p)` es la pila obtenida suprimiendo la cima de la pila `p`. Por ejemplo,

```
desapila p1  =>  2|3|-
```

```
desapila :: Pila a -> Pila a
desapila (P [])     = error "desapila la pila vacia"
desapila (P (_:xs)) = P  xs
```

# Implementación de las pilas mediante listas

- `(cima p)` es la cima de la pila `p`. Por ejemplo,

```
cima p1  ==  1
```

```
cima :: Pila a -> a
cima (P (x:_)) = x
cima (P [])    = error "cima de la pila vacia"
```

- `(desapila p)` es la pila obtenida suprimiendo la cima de la pila `p`. Por ejemplo,

```
desapila p1  =>  2|3|-
```

```
desapila :: Pila a -> Pila a
desapila (P [])     = error "desapila la pila vacia"
desapila (P (_:xs)) = P  xs
```

# Implementación de las pilas mediante listas

- `(esVacia p)` se verifica si `p` es la pila vacía. Por ejemplo,

```
esVacia p1     ==  False
esVacia vacia  ==  True
```

```
esVacia :: Pila a -> Bool
esVacia (P xs) = null xs
```

# Implementación de las pilas mediante listas

- `(esVacia p)` se verifica si `p` es la pila vacía. Por ejemplo,

```
esVacia p1     ==  False
esVacia vacia  ==  True
```

```
esVacia :: Pila a -> Bool
esVacia (P xs) = null xs
```

# Tema 14: El TAD de las pilas

# Importación de librerías

- Importación de la implementación de pilas que se desea comprobar.

```
import PilaConTipoDeDatoAlgebraico
-- import PilaConListas
```

- Importación de las librerías de comprobación

```
import Test.QuickCheck
import Test.Framework
import Test.Framework.Providers.QuickCheck2
```

# Tema 14: El TAD de las pilas

# Generador de pilas

- `genPila` es un generador de pilas. Por ejemplo,

```
ghci> sample genPila
0|0|-
-6|4|-3|3|0|-
-
9|5|-1|-3|0|-8|-5|-7|2|-
...
```

```
genPila :: (Num a, Arbitrary a) => Gen (Pila a)
genPila = do xs <- listOf arbitrary
             return (foldr apila vacia xs)

instance (Arbitrary a, Num a) => Arbitrary (Pila a) where
    arbitrary = genPila
```

# Tema 14: El TAD de las pilas

# Especificación de las propiedades de pilas

- La cima de la pila que resulta de añadir `x` a la pila `p` es `x`.

```
prop_cima_apila :: Int -> Pila Int -> Bool
prop_cima_apila x p =
    cima (apila x p) == x
```

- La pila que resulta de desapilar después de añadir cualquier elemento a una pila `p` es `p`.

```
prop_desapila_apila :: Int -> Pila Int -> Bool
prop_desapila_apila x p =
    desapila (apila x p) == p
```

# Especificación de las propiedades de pilas

- La cima de la pila que resulta de añadir `x` a la pila `p` es `x`.

```
prop_cima_apila :: Int -> Pila Int -> Bool
prop_cima_apila x p =
    cima (apila x p) == x
```

- La pila que resulta de desapilar después de añadir cualquier elemento a una pila `p` es `p`.

```
prop_desapila_apila :: Int -> Pila Int -> Bool
prop_desapila_apila x p =
    desapila (apila x p) == p
```

# Especificación de las propiedades de pilas

- La cima de la pila que resulta de añadir `x` a la pila `p` es `x`.

```
prop_cima_apila :: Int -> Pila Int -> Bool
prop_cima_apila x p =
    cima (apila x p) == x
```

- La pila que resulta de desapilar después de añadir cualquier elemento a una pila `p` es `p`.

```
prop_desapila_apila :: Int -> Pila Int -> Bool
prop_desapila_apila x p =
    desapila (apila x p) == p
```

# Especificación de las propiedades de pila

- La pila vacía está vacía.

```
prop_vacia_esta_vacia :: Bool
prop_vacia_esta_vacia =
    esVacia vacia
```

- La pila que resulta de añadir un elemento en un pila cualquiera no es vacía.

```
prop_apila_no_es_vacia :: Int -> Pila Int -> Bool
prop_apila_no_es_vacia x p =
    not (esVacia (apila x p))
```

# Especificación de las propiedades de pila

- La pila vacía está vacía.

```
prop_vacia_esta_vacia :: Bool
prop_vacia_esta_vacia =
    esVacia vacia
```

- La pila que resulta de añadir un elemento en un pila cualquiera no es vacía.

```
prop_apila_no_es_vacia :: Int -> Pila Int -> Bool
prop_apila_no_es_vacia x p =
    not (esVacia (apila x p))
```

# Especificación de las propiedades de pila

- La pila vacía está vacía.

```
prop_vacia_esta_vacia :: Bool
prop_vacia_esta_vacia =
    esVacia vacia
```

- La pila que resulta de añadir un elemento en un pila cualquiera no es vacía.

```
prop_apila_no_es_vacia :: Int -> Pila Int -> Bool
prop_apila_no_es_vacia x p =
    not (esVacia (apila x p))
```

# Tema 14: El TAD de las pilas

# Definición del procedimiento de comprobación

- `compruebaPropiedades` comprueba todas las propiedades con la plataforma de verificación.

```
compruebaPropiedades =
    defaultMain
        [testGroup "Propiedades del TAD pilas"
          [testProperty "P1" prop_cima_apila,
           testProperty "P2" prop_desapila_apila,
           testProperty "P3" prop_vacia_esta_vacia,
           testProperty "P4" prop_apila_no_es_vacia]]
```

# Comprobación de las propiedades de las pilas

```
ghci> compruebaPropiedades
Propiedades del TAD pilas:
  P1: [OK, passed 100 tests]
  P2: [OK, passed 100 tests]
  P3: [OK, passed 100 tests]
  P4: [OK, passed 100 tests]

         Properties  Total
 Passed  4           4
 Failed  0           0
 Total   4           4
```