Especificación e Implementación de TADs: Conjuntos Acotados

Cátedra AED

DC-UBA

2 cuat, 2023

Outline

Outline

Repasos: TADs

- Nos interesan las soluciones a problemas recurrentes de organización de información

Repasos: TADs

- Nos interesan las soluciones a problemas recurrentes de organización de información
- Esas soluciones van a ser las “estructuras de datos” y sus algoritmos de manipulación asociados (2da parte de la materia)

Repasos: TADs

- Nos interesan las soluciones a problemas recurrentes de organización de información
- Esas soluciones van a ser las "estructuras de datos" y sus algoritmos de manipulación asociados (2da parte de la materia)
- La presentación abstracta de los problemas (en forma de interface y comportamiento esperado) son los Tipos Abstractos de Datos

Repasos: TADs

- Nos interesan las soluciones a problemas recurrentes de organización de información
- Esas soluciones van a ser las "estructuras de datos" y sus algoritmos de manipulación asociados (2da parte de la materia)
- La presentación abstracta de los problemas (en forma de interface y comportamiento esperado) son los Tipos Abstractos de Datos
- Esta presentación abstracta facilita el buen uso y el razonamiento sobre estructuras de datos

Objetivos de esta clase

- Repasar punta a punta los conceptos de especificación e implementación de TADs

Objetivos de esta clase

- Repasar punta a punta los conceptos de especificación e implementación de TADs
- Recalcar el "information hidding"

Objetivos de esta clase

- Repasar punta a punta los conceptos de especificación e implementación de TADs
- Recalcar el "information hidding"
- Ver manipulación de pre y post para que sea simple entender la correctitud o derivación de la implementación. Ilustrando la idea de fortalecer post y debilitar pre (a.k.a., Principio de Sustitución de Liskov, behavioral subtyping, design by contract, etc.)

Outline

1. Introducción
2. Especificando TAD Conjunto Acotado
3. Implementación BoundedSet sobre Vector de Bits

Valores Conceptuales

- Necesitamos un tipo Conceptual (matemático) para los observadores
- En este caso, Conjunto de Naturales

Presentación

`TAD BoundedSet`

- `Obs set: Conj[`$\mathbb{N}$`]`
- `Obs bou:` $\mathbb{N}$ Tendrá un uso formal en la especificación: tecnicamente, es una "variable de historia" que permitirá justamente saber con qué cota se creo el conjunto y mostrar que no cambia.
  Veamos las operaciones...

Presentación

TAD BoundedSet

- Obs set: Conj[$\mathbb{N}$]
- Obs bou: $\mathbb{N}$ Tendrá un uso formal en la especificación: tecnicamente, es una "variable de historia" que permitirá justamente saber con qué cota se creo el conjunto y mostrar que no cambia.
  Veamos las operaciones...
- **Proc EmptySet (in bound: $\mathbb{N}$): BoundedSet**
  asegura res.set = $\emptyset$ $\wedge$ res.bou = bound
- **Proc Add (inout s: BoundedSet ,in d: $\mathbb{N}$)**
  requiere d $\leq$ s.bou $\wedge$ s=s_0
  asegura s.set = s_0.set $\cup\{$d$\}$ $\wedge$ s.bou=s_0.bou
- **Proc In? (in s: BoundedSet, in d: $\mathbb{N}$): Bool**
  asegura (res = True) $\Leftrightarrow$ (d $\in$ s.set)

Outline

Representación

- ¿Qué estructura puede proponer el desarrollador del TAD?
- ¿Qué contrato funcional -en términos de esa estructura- debería cumplir cada operación para decir que es una implementación correcta del TAD?
- El invariante de representación y la función de abstracción son los conceptos claves (¡aunque no se implementan realmente!)
- Nada de esto es visible al que usa el TAD (el código "cliente"). Ese es el principio de "information hidding" que simplifica el cambio y el razonamiento para el código cliente de este módulo

Representación

- ¿Qué estructura puede proponer el desarrollador del TAD?
- ¿Qué contrato funcional -en términos de esa estructura- debería cumplir cada operación para decir que es una implementación correcta del TAD?
- El invariante de representación y la función de abstracción son los conceptos claves (¡aunque no se implementan realmente!)
- Nada de esto es visible al que usa el TAD (el código "cliente"). Ese es el principio de "information hidding" que simplifica el cambio y el razonamiento para el código cliente de este módulo

```
Módulo BSVB implementa BoundedSet {
 arr:  Array<Bool>
```

Nota: BSBV se lee Bounded Set Using Bit Vector y es una estructura con un solo campo `arr` que es un arreglo de booleanos

Invariante de Representación

Invariante de Representación

```
pred InvRep(bsbv:  BSBV)
```

```
{def(bsbv.arr)}
```

Nota: cuando hablemos de memoria veremos exactemente qué es este def.

Función de Abstracción

Función de Abstracción

α(bsbv:BSVB):boundedSet
Definida en términos de los observadores del TAD

Función de Abstracción

α(bsbv:BSVB):boundedSet
Definida en términos de los observadores del TAD

α(bsbv).bou = (bsvb.arr.length)-1 $\wedge$
α(bsbv).set = $\{$n:$\mathbb{N}$|n$\leq$(bsvb.arr.length)-1 $\wedge_l$ bsbv[n]=True$\}$

Especificación Operaciones sobre la Estructura

```
Proc EmptySet (in bound: ℕ): BSBV
```

Especificación Operaciones sobre la Estructura

Proc EmptySet (in bound: $\mathbb{N}$): BSBV

requiere TRUE

Proc EmptySet (in bound: $\mathbb{N}$): BSBV

requiere TRUE

asegura InvRep(res) $\wedge_l$ α(res).set=$\emptyset$ $\wedge$ α(res).bou=bound

Especificación Operaciones sobre la Estructura

Proc EmptySet (in bound: $\mathbb{N}$): BSBV

requiere TRUE

asegura InvRep(res) $\wedge_l$ α(res).set=$\emptyset$ $\wedge$ α(res).bou=bound

(principio de sustitución: la postcondición puede hacerse más fuerte)

$\Leftarrow$ (x def. α e InvRep)

asegura def(res.arr) $\wedge_l$ ($\forall$ n:$\mathbb{N}$. n$\leq$(res.arr.length)-1 $\Rightarrow_l$ res[n]=False) $\wedge$ res.arr.length=bound+1

Código Operaciones sobre la Estructura

```
Proc EmptySet (in bound: ℕ): BSBV
```

Código Operaciones sobre la Estructura

Proc EmptySet (in bound: $\mathbb{N}$): BSBV

requiere TRUE

```
var aux:BSBV;
aux:=New(BSBV);
aux.arr:=NewArray<Bool>(bound+1);
RETURN aux
```

asegura def(res.arr) $\wedge_l$ ($\forall$ n:$\mathbb{N}$. n$\leq$(res.arr.length)-1 $\Rightarrow_l$ res[n]=False) $\wedge$ res.arr.length=bound+1

True $\Rightarrow$ Wp(asegura, código) por semántica axiomática asumida del New de Array (crea un arreglo de capacidad establecida y lo setea todo en False). OJO(S): (1) asumimos que siempre hay lugar en la memoria (2) esto puede ser caro en términos de tiempo: "O(bound)". Más adelante vamos a entender formalmente esto

Especificación Operaciones sobre la Estructura

Proc Add (inout bsbv: BSBV, in d: $\mathbb{N}$)

Especificación Operaciones sobre la Estructura

Proc Add (inout bsbv: BSBV, in d: $\mathbb{N}$)

requiere InvRep(bsbv) $\wedge_l$ (d $\leq$ α(bsbv).bou) $\wedge$ bsbv_0=bsbv

(principio de sustitución: la precondicion puede debilitarse)

$\Rightarrow$

Especificación Operaciones sobre la Estructura

Proc Add (inout bsbv: BSBV, in d: $\mathbb{N}$)

requiere InvRep(bsbv) $\wedge_l$ (d $\leq$ α(bsbv).bou) $\wedge$ $bsbv_0$=bsbv

(principio de sustitución: la precondicion puede debilitarse)

$\Rightarrow$

requiere def(bsbv.arr) $\wedge_l$ (d$\leq$bsbv.arr.length-1) $\wedge$ d=d_0 $\wedge$ $bsbv_0$=bsbv

asegura InvRep(bsbv) $\wedge_l$ d=d_0 $\wedge$ α(bsbv).set=α($bsbv_0$).set $\cup\{$d$\}$ $\wedge$ α(bsbv).bou=α($bsbv_0$).bou

Especificación Operaciones sobre la Estructura

Proc Add (inout bsbv: BSBV, in d: $\mathbb{N}$)

requiere InvRep(bsbv) $\wedge_l$ (d $\leq$ α(bsbv).bou) $\wedge$ $bsbv_0$=bsbv

(principio de sustitución: la precondicion puede debilitarse)

$\Rightarrow$

requiere def(bsbv.arr) $\wedge_l$ (d$\leq$bsbv.arr.length-1) $\wedge$ d=d_0 $\wedge$ $bsbv_0$=bsbv

asegura InvRep(bsbv) $\wedge_l$ d=d_0 $\wedge$ α(bsbv).set=α($bsbv_0$).set $\cup\{$d$\}$ $\wedge$ α(bsbv).bou=α($bsbv_0$).bou

Fortaleciendo la post y def. de α

$\Leftarrow$

Especificación Operaciones sobre la Estructura

Proc Add (inout bsbv: BSBV, in d: $\mathbb{N}$)

requiere InvRep(bsbv) $\wedge_l$ (d $\leq$ α(bsbv).bou) $\wedge$ bsbv_0=bsbv

(principio de sustitución: la precondicion puede debilitarse)

$\Rightarrow$

requiere def(bsbv.arr) $\wedge_l$ (d$\leq$bsbv.arr.length-1) $\wedge$ d=d_0 $\wedge$ bsbv_0=bsbv

asegura InvRep(bsbv) $\wedge_l$ d=d_0 $\wedge$ α(bsbv).set=α(bsbv_0).set $\cup$\{d\} $\wedge$ α(bsbv).bou=α(bsbv_0).bou

Fortaleciendo la post y def. de α

$\Leftarrow$

asegura def(bsbv.arr) $\wedge_l$ d=d_0 $\wedge$ (d $\leq$bsbv.arr.length-1) $\wedge_l$ bsbv.arr=SetAt(bsbv_0.arr,d,True) $\wedge$ bsbv.arr.length = bsbv_0.arr.length

Código Operaciones sobre la Estructura

Proc Add (inout bsbv: BSBV, in d: $\mathbb{N}$)

Código Operaciones sobre la Estructura

Proc Add (inout bsbv: BSBV, in d: $\mathbb{N}$)

requiere def(bsbv.arr) $\wedge_l$ (d$\leq$bsbv.arr.length-1) $\wedge$ d=d_0 $\wedge$ $bsbv_0$=bsbv

```
bsbv.arr[d] := True;
RETURN
```

asegura def(bsbv.arr) $\wedge_l$ d=d_0 $\wedge$ (d $\leq$bsbv.arr.length-1) $\wedge_l$ bsbv.arr=SetAt($bsbv_0$.arr,d,True)

Notas: el "requiere" implica Wp(asegura, código) x semántica de la asignación de array (el índice tiene que estar en rango y lo que hace es cambiar el valor en la posición del índice sin alterar la capacidad).
Se podría hacer una versión con Precondición True (aún más débil) con chequeo d sea menor o igual que capacity. El código sería más robusto. Es lo que típicamente se intenta hacer -cuando se puede- en las bibliotecas y se informa con algún mecanismo la violación de precondiciones

Especificación Operaciones sobre la Estructura de Datos

`Proc In? (in bsbv: BSBV, in d:` $\mathbb{N}$`): Bool`

Proc In? (in bsbv: BSBV, in d: $\mathbb{N}$): Bool

requiere InvRep(bsbv) $\wedge$ $bsbv_0$=bsbv

requiere def(bsbv.arr) $\wedge$ $bsbv_0$=bsbv

Proc In? (in bsbv: BSBV, in d: $\mathbb{N}$): Bool

requiere InvRep(bsbv) $\wedge$ $bsbv_0$=bsbv

requiere def(bsbv.arr) $\wedge$ $bsbv_0$=bsbv

asegura $bsbv_0$=bsbv $\wedge$ InvRep(bsbv) $\wedge_l$ ((res=True) $\Leftrightarrow$ (d$\in\alpha$(bsbv).set))

Especificación Operaciones sobre la Estructura de Datos

Proc In? (in bsbv: BSBV, in d: $\mathbb{N}$): Bool

requiere InvRep(bsbv) $\wedge$ $\texttt{bsbv}_0$=bsbv

requiere def(bsbv.arr) $\wedge$ $\texttt{bsbv}_0$=bsbv

asegura $\texttt{bsbv}_0$=bsbv $\wedge$ InvRep(bsbv) $\wedge_l$ ((res=True) $\Leftrightarrow$ (d$\in\alpha$(bsbv).set))

Fortaleciendo la post

$\Leftarrow$

asegura def(bsbv.arr) $\wedge$ $\texttt{bsbv}_0$=bsbv $\wedge_l$ ((res=True) $\Leftrightarrow$ (d$\leq$bsbv.arr.length-1 $\wedge_l$ bsbv.arr[d]=True))

Código Operaciones sobre la Estructura de Datos

Proc In? (in bsbv: BSBV, in d: $\mathbb{N}$): Bool

Código Operaciones sobre la Estructura de Datos

Proc In? (in bsbv: BSBV, in d: $\mathbb{N}$): Bool

requiere def(bsbv.arr) $\wedge$ $bsbv_0$=bsbv

```
IF d ≤ (bsbv.arr.length)-1
    THEN
    res := bsbv.arr[d]
     ELSE
    res := False
ENDIF
RETURN res
```

asegura def(bsbv.arr) $\wedge$ $bsbv_0$=bsbv $\wedge_l$ ((res=True) $\Leftrightarrow$ (d$\leq$bsbv.arr.length-1 $\wedge_l$ bsbv.arr[d]=True))