

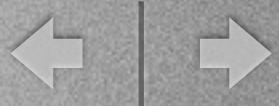


Algoritmos y estructuras de datos II

Diseño jerárquico de tipos

Carlos Gustavo Lopez Pombo
(Charlie)

Departamento de Computación,
Facultad de ciencias exactas y naturales,
Universidad de Buenos Aires



¿Qué significa diseñar?

- Pasar de la descripción del **qué** del problema al **cómo**
- Considerar aspectos no funcionales, por ejemplo, eficiencia en tiempo y espacio de acuerdo al contexto de uso
- Permitir un cambio de paradigma (del utilizado para especificar al utilizado para programar) que resulte “ordenado”.



¿Qué significa diseñar?

En nuestro caso se trata de producir una implementación en el paradigma imperativo de un tipo abstracto de datos descripto en un lenguaje de características lógicas (en particular funcional).

ConjuntoDeNat → Conj[nat]

Se explica con:



¿Qué significa diseñar?

En nuestro caso se trata de producir una implementación en el paradigma imperativo de un tipo abstracto de datos descripto en un lenguaje de características lógicas (en particular funcional).

ConjuntoDeNat → Conj[nat]

Se explica con:

Intuitivamente
¿Qué se resuelve en cada nivel?



¿Qué significa jerárquico?

Que pensaremos la resolución del cómo a partir de representaciones de un tipo sobre otros separando responsabilidades en la construcción de la solución

conjuntoDeNat

↓ Se representa con:

secuenciaDeNat

↓ Se representa con:

⋮

↓ Se representa con:

punteros



¿Qué significa jerárquico?

Que pensaremos la resolución del cómo a partir de representaciones de un tipo sobre otros separando responsabilidades en la construcción de la solución

conjuntoDeNat

↓ Se representa con:

secuenciaDeNat

↓ Se representa con:

⋮

↓ Se representa con:

punteros

Intuitivamente

¿Qué se resuelve en cada nivel?



Lenguaje de diseño

Hay cuatro cosas a tener en cuenta, que son de relieve antes de comenzar a diseñar, por sus consecuencias sobre el comportamiento de nuestras implementaciones:

- Tipos de datos
- Declaración de operaciones
- Pasaje de parámetros
- Asignación



Métodología de diseño

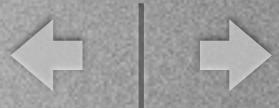
Desde un punto de vista abstracto diseñar implica las siguientes tareas:

- Elección del tipo a diseñar
- Introducción de los elementos no funcionales
- Vinculación entre la representación y su abstracción
- Iteración sobre los tipos restantes



Métodología de diseño

- Los **aspectos de la interfaz** de un tipo describe todo elemento referido a los aspectos de uso de dicho tipo, es decir, toda cuestión que resulte visible desde “afuera”,
- Las **pautas de implementación** serán todo aspecto que refiera a cuestiones vinculadas a los medios a través de los cuales el tipo garantiza esos aspectos de uso.



Métodología de diseño

La definición de la **interfaz** de un módulo de diseño implica tomar cuenta de varias cosas... esencialmente debe explicarle al eventual usuario todos los aspectos relativos a los servicios que exporta.



Métodología de diseño

La definición de la **interfaz** de un módulo de diseño implica tomar cuenta de varias cosas... esencialmente debe explicarle al eventual usuario todos los aspectos relativos a los servicios que exporta.

Servicios exportados: describe para cada operación su orden de complejidad, aspectos de aliasing, efectos colaterales sobre los argumentos, etc.

Interfaz: define en el paradigma imperativo las operaciones exportadas junto con su precondición y postcondición. Esto establecerá la relación entre la implementación de las operaciones y su especificación.



Ejemplo

TAD CONJ(NAT)

usa BOOL, NAT

géneros conj(nat)

exporta conj(nat), $\bullet \in \bullet, \emptyset, Ag, \emptyset?$, mínimo

observadores básicos

$\bullet \in \bullet : \text{nat} \times \text{conj}(\text{nat}) \rightarrow \text{bool}$

generadores

\emptyset : $\longrightarrow \text{conj}(\text{nat})$

Ag : $\text{nat} \times \text{conj}(\text{nat}) \rightarrow \text{conj}(\text{nat})$

otras operaciones

$$\bullet - \{\bullet\} : \text{conj(nat)} \times \text{nat} \longrightarrow \text{conj(nat)}$$

$\emptyset?$: conj(nat) \longrightarrow bool

mínimo : conj(nat) c → nat

$$\{\neg\emptyset?(c)\}$$

axiomas los tradicionales

Fin TAD



Métodología de diseño

Contexto de uso:

- la obtención del mínimo debe tener complejidad $O(1)$
- quitar un elemento debe tener complejidad $O(n)$ con n la cantidad de elementos agregados al conjunto
- agregar un elemento debe tener complejidad $O(1)$



Ejemplo

Servicios exportados:

pertenecে: no produce aliasing ni efectos colaterales sobre los argumentos, posee orden de complejidad temporal $O(n)$ con n la cantidad de elementos agregados al conjunto,

vacio: no produce aliasing ni efectos colaterales, posee orden de complejidad temporal $O(1)$

agregar: no produce aliasing, modifica colateralmente el conjunto argumento, posee orden de complejidad temporal $O(1)$,

quitar: no produce aliasing sobre los argumentos, modifica colateralmente el conjunto argumento, posee orden de complejidad temporal $O(n)$ con n la cantidad de elementos agregados al conjunto,

vacio?: no produce aliasing ni efectos colaterales, posee orden de complejidad temporal $O(1)$

minimo: no produce aliasing ni efectos colaterales, posee orden de complejidad temporal $O(1)$



Métodología de diseño

Relación entre los valores imperativos y los valores lógicos

Interfaz:

Quiséramos poder decir lo siguiente...

```
pertenece (in conjuntoDeNat c, in nat n) -> res: bool
{true}
{res =obs n pertenece c}
```

¿Qué problema observan?



Métodología de diseño

Relación entre los valores imperativos y los valores lógicos

Si el lenguaje de implementación es diferente del lenguaje de especificación,

¿qué diferencia existe entre los valores que pueden tomar las variables imperativas respecto de los términos lógicos?

¿cómo podemos vincularlos?



Ejemplo

Para resolver este problema introduciremos una función que para cada valor imperativo nos retornará el término lógico al cual representa, de forma que este pueda participar de los predicados lógicos que definen el comportamiento formal de las operaciones...

... es decir que introduciremos una notación que vincule un género imperativo GI con el género GT correspondiente a su especificación,

$$\hat{\bullet} : GI \rightarrow GT$$



Métodología de diseño

Relación entre los valores imperativos y los valores lógicos

Interfaz:

Es decir que escribiremos...

pertenece (in c: conjuntoDeNat, in n: nat) -> res: bool

{true}

{ $\widehat{\text{res}} = \text{obs} \wedge \widehat{n} \text{ pertenece } \widehat{c}$ }



Ejemplo

Interfaz:

pertenece (in c: conjuntoDeNat, in n: nat) -> res: bool

{true}

{ $\widehat{\text{res}}$ =obs \widehat{n} pertenece \widehat{c} }

vacio () -> res: conjuntoDeNat

{true}

{ $\widehat{\text{res}}$ =obs vacio }

agregar (in/out c: conjuntoDeNat, in n: nat)

{ \widehat{c} =obs c_0 }

{ \widehat{c} =obs Ag (c_0 , \widehat{n}) }

quitar (in/out c: conjuntoDeNat c, in n: nat)

{ \widehat{c} =obs c_0 }

{ \widehat{c} =obs c_0 - { \widehat{n} } }



Ejemplo

Interfaz:

vacio? (**in** c: conjuntoDeNat) -> res: **bool**

{**true**}

{ $\widehat{\text{res}} = \text{obs vacio? } (\widehat{c})$ }

minimo(**in** c: conjuntoDeNat) -> res: **nat**

{*not vacio? (c)*}

{ $\widehat{\text{res}} = \text{obs minimo } (\widehat{c})$ }



Métodología de diseño

La definición de la **representación** de un módulo de diseño implica tomar en cuenta todo aspecto referido a cómo se satisfacen los requerimientos declarados en la interfaz.



Métodología de diseño

La definición de la **representación** de un módulo de diseño implica tomar en cuenta todo aspecto referido a cómo se satisfacen los requerimientos declarados en la interfaz.

Estructura: describe la estructura interna sobre la cual las operaciones aplican.

Relación entre la representación y la abstracción: por un lado expone toda restricción sobre la estructura de representación a fin de que efectivamente pueda ser considerada una implementación de un valor del tipo al que implementa; y por otro vincula los valores con su contraparte abstracta, es decir, con el término de la especificación que corresponda.

Algoritmos: ... sí, es eso, la programación en pseudo-código de las operaciones, tanto las exportadas como las auxiliares, y para todos ellos incluye el cálculo detallado que justifica su complejidad

Servicios usados: declara toda demanda de complejidad, aliasing o efecto colateral que los servicios usados en la programación de los algoritmos deba satisfacer.



Métodología de diseño

Estructura de representación

La **estructura de representación** describe los valores sobre los cuales se representará el género que se está implementando.

Esta tiene que tener en cuenta la posibilidad, no sólo de ser capaz de representar **todos** los términos del género de la especificación sino también que las operaciones sean implementables de acuerdo a las exigencias del contexto de uso.



Ejemplo

Estructura de representación:

conjuntoDeNat se representa con secuenciaDeNat

¿Alcanza? Claramente todos los conjuntos de naturales pueden representarse con secuencias de naturales, no?

¿Qué nos está faltando? Repasemos las operaciones para ver si esta estructura nos permite su implementación dentro del contexto de uso propuesto...



Ejemplo

Estructura de representación:

conjuntoDeNat se representa con <secuenciaDeNat, nat>

¡Ahora está mejor! ¿Cómo haríamos si no para obtener el mínimo en $O(1)$?



Ejemplo

Estructura de representación:

conjuntoDeNat se representa con <secuenciaDeNat, nat>

¡Ahora está mejor! ¿Cómo haríamos si no para obtener el mínimo en $O(1)$?

¿Suficiente?



Ejemplo

Estructura de representación:

conjuntoDeNat se representa con <secuenciaDeNat, nat>

¡Ahora está mejor! ¿Cómo haríamos si no para obtener el mínimo en $O(1)$?

¿Suficiente?

Sí, al menos por ahora...



Métodología de diseño

Estructura de representación

Ya estamos bastante convencidos de que esta estructura permitirá representar todos los términos del género de la especificación.

La pregunta que cabe ahora es si toda estructura es legítima como representación de un término de la especificación... ¿qué les parece?

conjuntoDeNat se representa con <secuenciaDeNat, nat>



Métodología de diseño

Invariante de representación

El **invariante de representación** nos ayuda en la tarea de filtrar toda aquella estructura que no implementa un término de la especificación.

Es decir que de alguna forma expresa un predicado de “sanidad” de la estructura...

¿ $\langle [1,2,5,8,9,15], 7 \rangle$ es legítima? ¿Por qué?

¿ $\langle [1,2,5,8,9,15], 5 \rangle$ es legítima? ¿Por qué?

¿ $\langle [1,2,2,5,8,8,9,15,15], 1 \rangle$ es legítima? ¿Por qué?



Ejemplo

$Rep : \langle \widehat{\text{secuenciaDeNat}}, \text{nat} \rangle \rightarrow \text{bool}$

$(\forall \langle s, n \rangle : \text{secu[nat]} \times \text{nat})(Rep(\langle s, n \rangle) \iff \text{vacia?}(s) \vee (\text{esta?}(s, n) \wedge (\forall n' : \text{nat})(\text{esta?}(s, n') \wedge \neg n =_{obs} n' \implies n < n')) \wedge \text{sinRepetidos}(s))$

$\text{esta?} : \text{secu[nat]} \times \text{nat} \rightarrow \text{bool}$

$\text{esta;}([], n) \equiv \text{false}$

$\text{esta?}(a \bullet s, n) \equiv a = n \vee \text{esta?}(s, n)$

$\text{sinRepetidos} : \text{secu[nat]} \rightarrow \text{bool}$

$\text{sinRepetidos}([]) \equiv \text{true}$

$\text{sinRepetidos}(a \bullet s) \equiv \neg \text{esta?}(a, s) \wedge \text{sinRepetidos}(s)$

Si necesitáramos que la secuencia no tenga repetidos. ¡Ojo, para otro contexto de uso completamente distinto!



Métodología de diseño

Invariante de representación

El sentido que se le otorga al invariante de representación es que cada operación requiere para su ejecución que el invariante de representación valga sobre las estructuras pasadas como parámetro y si esto se cumple, entonces esta asegura que también vale al finalizar la ejecución...

¿A qué les suena?



Métodología de diseño

Invariante de representación

El sentido que se le otorga al invariante de representación es que cada operación requiere para su ejecución que el invariante de representación valga sobre las estructuras pasadas como parámetro y si esto se cumple, entonces esta asegura que también vale al finalizar la ejecución...

¿A qué les suena?

¡Sí señores, la satisfacción del invariante está presente en forma implícita en la pre y la post de todas las operaciones exportadas!



Métodología de diseño

Función de abstracción

La **función de abstracción** es una herramienta que permite vincular una estructura con el valor abstracto al que representa. ¿Cómo les parece que esto puede hacerse? Es decir, ¿qué debemos usar para caracterizar el término abstracto que representa a una estructura particular?



Métodología de diseño

Función de abstracción

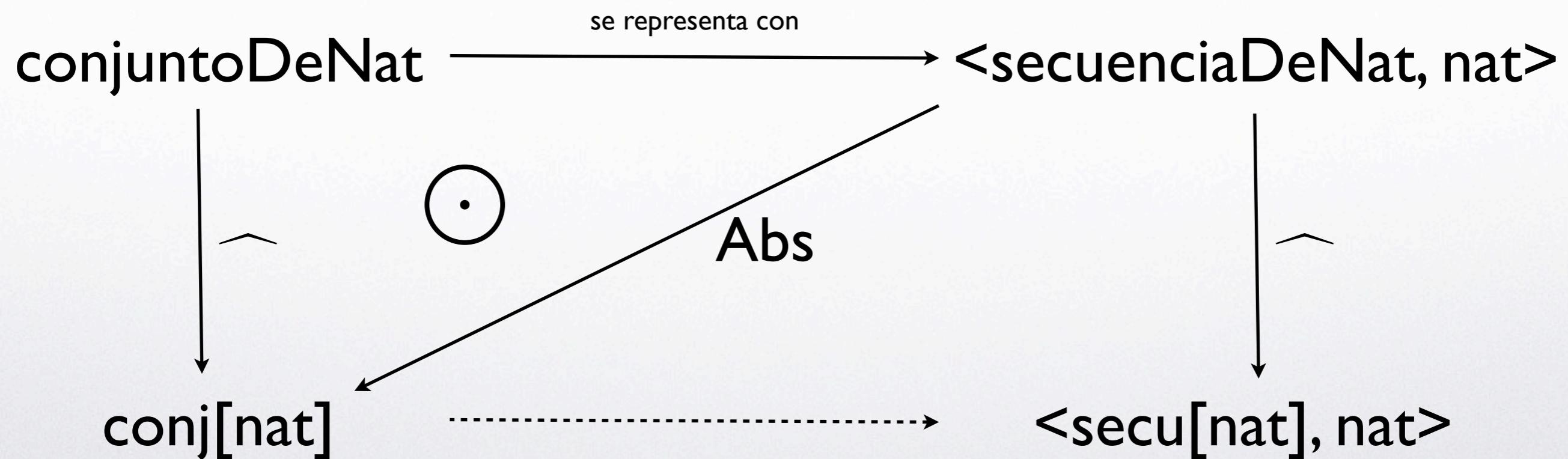
La **función de abstracción** es una herramienta que permite vincular una estructura con el valor abstracto al que representa. ¿Cómo les parece que esto puede hacerse? Es decir, ¿qué debemos usar para caracterizar el término abstracto que representa a una estructura particular?

¡Así es, la manera en la que se caracteriza un término es o bien a través de los generadores o de los observadores! Normalmente el uso de observadores resulta más sencillo



Métodología de diseño

Función de abstracción





Ejemplo

$Abs : \langle s, n \rangle : \langle \widehat{\text{secuenciaDeNat}}, \text{nat} \rangle \rightarrow \text{conj}[\text{nat}] \quad Rep(\langle s, n \rangle)$
 $(\forall \langle s, n \rangle : \langle \text{secu}[\text{nat}], \text{nat} \rangle)(Abs(\langle s, n \rangle) =_{obs} c \mid$
 $\quad (\forall n' : \text{nat})(esta?(s, n') \iff n' \in c))$

$esta? : \text{secu}[\text{nat}] \times \text{nat} \rightarrow \text{bool}$

$esta?([], n) \equiv \text{false}$

$esta?(a \bullet s, n) \equiv (a = n) \vee esta?(s, n)$



Métodología de diseño

Algoritmos

iQuitar (in/out c: <secuencia: secuenciaDeNat, minimo: nat>, in n: nat)

```
if (!vacia?(c.secuencia)) then
    secuenciaDeNat s = vacia(), s' = c.secuencia;
    while (!vacia?(s')) do
        if (prim(s') != n) then
            puntito (s, prim(s'))
            s' = resto (s')
        int min = prim(s)
        Buscar el mínimo de s y colocarlo en min...
        c.secuencia = s
        c.minimo = min
```



Métodología de diseño

Algoritmos

iQuitar (in/out c: <secuencia: secuenciaDeNat, minimo: nat>, in n: nat)

```
if (!vacia?(c.secuencia)) then          O(1)
    secuenciaDeNat s = vacia(), s' = c.secuencia;
    while (!vacia?(s')) do               m*O(1)
        if (prim(s') != n) then          O(1)
            puntito (s, prim(s'))      O(1)
            s' = resto (s')           O(1)
        int min = prim(s)             O(1)
        Buscar el mínimo de s y colocarlo en min... O(m)
        c.secuencia = s              O(1)
        c.minimo = min              O(1)
```



Métodología de diseño

Servicios usados

La operación vacia? debe tener complejidad $O(1)$ y no producir modificación alguna sobre los argumentos ni provocar efectos colaterales,

La operación puntito modifica el argumento a partir de producir el agregado del nuevo elemento y debe tener complejidad $O(1)$

...



¡Es todo por hoy!



¡Es todo por hoy!

