Introducción al diseño

Martin (March) Miguel

Departamento de Computación,
Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires

Algoritmos y Estructuras de Datos 2
2do cuatrimestre de 2016

Diferencias entre especificación y diseño

- En la etapa de especificación:
  - Nos ocupamos del '¿qué?'
  - Lo explicamos usando TADs
  - Ese '¿qué?' se explica bajo el paradigma funcional
- En la etapa de diseño:
  - Nos ocupamos del '¿cómo?'
  - Lo explicamos con *módulos de abstracción*
  - Ese '¿cómo?' se explica usando el paradigma imperativo

Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

- Interfaz: Es la sección accesible para los usuarios del módulo (que bien pueden ser otros módulos). Aquí se detallan los servicios exportados (principalmente las operaciones con su complejidad temporal y sus efectos colaterales).
- Representación: Esta sección no es accesible a los usuarios del módulo. Aquí se detalla la elección de estructura de representación y los algoritmos. Se justifica la elección de las estructuras así como la complejidad de los algoritmos.
- Servicios usados: Es la parte del módulo donde se detallan todas los supuestos sobre los servicios que exportan los otros módulos. Estos requisitos son los que justifican los análisis de complejidad que se hacen dentro de la sección Representación que a su vez justifican las promesas hechas en la interfaz.

Interfaz: Operaciones

Para cada operación que se exporta se debe declarar lo siguiente:

- Signatura
- Pre y postcondiciones
- Complejidad temporal
- Aliasing

Ejemplo: Conjunto en rango

TAD CONJUNTO EN RANGO

Ig. obs.: $(\forall c_1, c_2 : \text{conjran})(c_1 =_{\text{obs}} c_2 \Leftrightarrow (\text{low}(c_1) = \text{low}(c_2) \wedge \text{up}(c_1) = \text{up}(c_2) \wedge (\forall n : \text{nat})((n \in c_1) = (n \in c_2))))$

observadores básicos

$\bullet \in \bullet$: nat $\times$ conjran $\longrightarrow$ bool

low : conjran $\longrightarrow$ nat

up : conjran $\longrightarrow$ nat

generadores

$\emptyset$: nat l $\times$ nat u $\longrightarrow$ conjran $\qquad l \leq u$

Ag : nat n $\times$ conjran c $\longrightarrow$ conjran $\qquad \text{low}(c) \leq n \leq \text{up}(u)$

otras operaciones

$\#$: conjran $\longrightarrow$ nat

$\bullet - \{\bullet\}$: conjran $\times$ nat $\longrightarrow$ conjran

Fin TAD

Ejemplo: Conjunto en rango

TAD CONJUNTO EN RANGO

axiomas $\forall c : \text{conjran}, \forall l, u, n, n' : \text{nat}$

$$
\begin{array}{lcl}
n \in \emptyset(l, u) & \equiv & \text{false} \\
n \in \text{Ag}(n', c) & \equiv & (n = n') \vee (n \in c) \\
\text{low}(\emptyset(l, u)) & \equiv & \text{l} \\
\text{low}(\text{Ag}(n', c)) & \equiv & \text{low}(c) \\
\text{up}(\emptyset(l, u)) & \equiv & \text{u} \\
\text{up}(\text{Ag}(n', c)) & \equiv & \text{up}(c) \\
\#(\emptyset(l, u)) & \equiv & 0 \\
\#(\text{Ag}(n, c)) & \equiv & 1 + \#(c \text{ - } \{ \ n \ \} \ ) \\
\emptyset - \{n\} & \equiv & \emptyset \\
\text{Ag}(n', c) - \{n\} & \equiv & \textbf{if } n' = n \textbf{ then } c - \{n\} \textbf{ else } \text{Ag}(n', c - \{n\}) \textbf{ fi}
\end{array}
$$

Fin TAD

Interfaz: Encabezado

interfaz CONJENRANGO
usa: BOOL, NAT, ...
se explica con: Conjunto en Rango
género: `conjenrango`
operaciones:
...

Interfaz: Signatura

- Consideremos la siguiente operación del TAD:

  $\bullet \in \bullet \ : \ \text{nat} \times \text{conjran} \longrightarrow \text{bool}$

- Proponemos la siguiente signatura en la interfaz del diseño:

  Pertenece(in `n` : `nat`, in `c` : `conjenrango`) $\rightarrow$ `res` : `bool`

- Las variables `n`, `c` y `res` son del universo del diseño. Son instancias de los módulos correspondientes.

Interfaz: Pre y postcondiciones

- En la especificación, las operaciones de los TADs se explican a través de la axiomatización.
- El comportamiento de las operaciones en la interfaz se explica mediante pre y postcondiciones.

Pertenece(in `n` : `nat`, in `c` : `conjenrango`) $\rightarrow$ `res` : `bool`

{true} [Precondición]

$\{\widehat{\texttt{res}} = (\hat{\texttt{n}} \in \hat{\texttt{c}})\}$ [Postcondición]

- Las pre y postcondiciones se escriben usando las operaciones de TADs. Por eso aquí $\widehat{\texttt{res}}$, $\hat{\texttt{n}}$ y $\hat{\texttt{c}}$ son las instancias de los TADs Bool, Nat y Conjunto en Rango que se corresponden con las instancias `res`, `n` y `c` del mundo del diseño.
- Para no sobrecargarlos de formalidad, permitimos que omitan los sombreritos, pero hay que saber que hay una transformación implícita entre los mundos.

Interfaz: Operaciones low y up

- Para la operación del TAD

  $\text{low} : \text{conjran} \longrightarrow \text{nat}$

  podemos dar la siguiente declaración en la interfaz

  Lower(in c : `conjenrango`) $\rightarrow$ `res` : `nat`
  $\{\text{true}\}$
  $\{\widehat{\texttt{res}} = \text{low}(\hat{c})\}$

- Y para la operación del TAD

  $\text{up} : \text{conjran} \longrightarrow \text{nat}$

  daremos la siguiente declaración en la interfaz

  Upper(in c : `conjenrango`) $\rightarrow$ `res` : `nat`
  $\{\text{true}\}$
  $\{\widehat{\texttt{res}} = \text{up}(\hat{c})\}$

Interfaz: Operaciones $\emptyset$ y Ag

- Para la operación del TAD

  $\emptyset$: nat ℓ $\times$ nat u $\longrightarrow$ conjran $\qquad (\ell \leq u)$

  la declararemos en la interfaz como

  Vacío(in `l` : `nat`, in `u` : `nat`) $\rightarrow$ `res` : `conjenrango`

  $\{\hat{\mathtt{l}} \leq \hat{\mathtt{u}}\}$

  $\{\widehat{\mathtt{res}} = \emptyset(\hat{\mathtt{l}}, \hat{\mathtt{u}})\}$

- Y para la operación del TAD

  Ag : nat n $\times$ conjran c $\longrightarrow$ conjran

  $(\text{low}(c) \leq n \leq \text{up}(c))$

  la podemos declarar en la interfaz mediante

  Agregar(in `n` : `nat`, inout c : `conjenrango`)

  $\{\text{low}(\hat{\mathtt{c}}) \leq \hat{\mathtt{n}} \leq \text{up}(\hat{\mathtt{c}}) \wedge c_0 = \hat{\mathtt{c}}\}$

  $\{\hat{\mathtt{c}} = \text{Ag}(\hat{\mathtt{n}}, c_0)\}$

Interfaz: Operaciones # y −

- Para la operación del TAD

  $\# : \text{conjran} \longrightarrow \text{nat}$

  la declararemos en la interfaz como

  Cardinal(in c : `conjenrango`) → `res` : `nat`

  $\{true\}$

  $\{\widehat{\texttt{res}} = \#(\hat{c})\}$

- Y por último para la operación

  $\bullet - \{\bullet\} : \text{conjran} \times \text{nat} \longrightarrow \text{conjran}$

  la podemos declarar en la interfaz mediante

  Borrar(in `n` : `nat`, inout c : `conjenrango`)

  $\{c_0 = \hat{c}\}$

  $\{\hat{c} = c_0 - \{\hat{n}\}\}$

Interfaz: Complejidad y aspectos de aliasing

- La declaración de una función en una interfaz no está completa si no se explicitan la complejidad y los posibles efectos secundarios y de aliasing.
- Sin embargo no podemos completar esta información por el momento hasta que no sepamos como vamos a representar el tipo, cómo vamos a diseñar los algoritmos y qué supuestos haremos sobre los módulos auxiliares que utilizaremos.
- Teóricamente, las pre y post-condiciones también podrían depender de los algoritmos, pero en menor medida.

Representación

En la sección Representación:

- Se define la forma en que se representan las instancias del tipo que estamos diseño.
- Esta representación involucra la elección de una o más estructuras, la cual deberá estar debidamente justificada.
- Además aquí se especifican de manera precisa las relaciones entre la representación y la abstracción que representa.
- Se presentan los algoritmos y justifican las complejidades y efectos secundarios que se declararán en la interfaz.

TAD arreglo dimensionable

TAD ARREGLO DIMENSIONABLE(α)
Ig. obs.: ($\forall$ a, a' : ad(α)) (tam(a) $=_{obs}$ tam(a') $\wedge$
($\forall n$: nat)(($n <$ tam(a)) $\Longrightarrow$ $a[n] =_{obs} a'[n]$)))

observadores básicos

tam : ad(α) $\longrightarrow$ nat
$\bullet[\bullet]$: ad(α) a $\times$ nat n $\longrightarrow$ α $\quad n <$ tam(a)

generadores

crearArreglo : nat$\times\alpha$ $\longrightarrow$ ad(α)
$\bullet[\bullet] \leftarrow \bullet$: ad(α) a $\times$ nat n $\times$ α $\longrightarrow$ ad(α) $\quad n <$ tam(a)

axiomas $\forall$ ad(α) : a, $\forall$ α : e, $\forall$ nat : n, m n

tam(crearArreglo(n, v)) $\equiv$ n
tam(a [n] $\leftarrow$ e) $\equiv$ tam(a)
(crearArreglo(n, v)) [m] $\equiv$ v
(a [n] $\leftarrow$ e) [m] $\equiv$ **if** $n = m$ **then** e **else** a [m] **fi**

Fin TAD

- Vamos a suponer que tenemos un módulo de tipo *arrd* que se corresponde a este TAD.

Representación: Estructura de representación

- Elegimos la siguiente representación para el tipo:

> `conjenrango` **se representa con** `estr` donde
>
> `estr` es `tupla`$\langle$*lower* : `nat` $\times$
> *upper* : `nat` $\times$
> *elems* : `arrd`(`bool`)$\rangle$

  Aquí `tupla`, `nat` y `arrd` son tipos del diseño que se corresponden con los TADs Tupla, Nat y Arreglo Dimensionable.

- Esta descripción por sí sola no es suficiente. Para ser precisos debemos explicar a qué instancia del TAD Conjunto en Rango se corresponde cada instancia de `estr`.
- Para eso, debemos resolver primero el problema de indicar cuáles instancias de `estr` son 'admisible', es decir, son representaciones plausibles de alguna instancia del TAD.

Representación: Invariante de representación (Rep)

- El invariante de representación tiene la siguiente signatura:

$$\text{Rep} : \widehat{\texttt{estr}} \to \text{boolean}$$

  Aquellas instancias e del tipo $\widehat{\texttt{estr}}$ tales que Rep(e) es verdadera serán las representaciones 'admisibles', es decir, aquellas que bajo nuestra interpretación vamos a asignarle una determinada instancia del TAD que estamos diseñando.
- El invariante de representación es cierto en el momento inicial y final de cualquier operación que se le aplique a una instancia al módulo.

Representación: Invariante de representación (Rep)

- Como invariante de representación proponemos:

$$\begin{array}{l} \text{Rep} : \widehat{\texttt{estr}} \to \text{boolean} \\ (\forall e : \widehat{\texttt{estr}})(\text{Rep}(e) = \\ \quad (e.\mathit{lower} \leq e.\mathit{upper}) \wedge \\ (\mathit{tam}(e.\mathit{elems}) \equiv e.\mathit{upper} - e.\mathit{lower} + 1)) \end{array}$$

Representación: Función de abstracción (Abs)

- Está definida para cada instancia $e : \widehat{\texttt{estr}}$ que hace verdadera la invariante de representación y le asigna la instancia correspondiente del TAD que estamos diseñando.
- Por lo tanto la signatura de la función de abstracción es

$$\text{Abs} : \widehat{\texttt{estr}}\ e \to \textit{tipo\_abstracto\_representado} \qquad (\text{Rep}(e))$$

- Para el caso del TAD Conjunto en Rango proponemos

$$\text{Abs} : \widehat{\texttt{estr}}\ e \to \text{conjran}$$

$$\begin{aligned}\text{Abs}(e) = c \ / \ &((\text{low}(c) = e.\textit{lower}) \wedge (\text{up}(c) = e.\textit{upper}) \wedge_L \\ &(\forall n : \text{nat})(n \in c \Leftrightarrow ((\text{low}(c) \leq n \leq \textit{up}(c)) \\ &\wedge_L e.\textit{elems}[n - \textit{low}(c)])))\end{aligned}$$

- Notemos que la función Abs(e) está bien definida gracias a tener a Rep(e) como precondición.

Representación: Justificación de la estructura de representación

- La elección de la estructura debe justificarse debidamente dentro de esta sección.
- Por ahora no nos vamos a ocupar de esto.

Representación: Algoritmo para Pertenece

- En la interfaz habíamos declarado la operación:

  Pertenece(in $\texttt{n} : \texttt{nat}$, in $\texttt{c} : \texttt{conjenrango}$) $\rightarrow \texttt{res} : \texttt{bool}$

  $\{\text{true}\}$

  $\{\widehat{\texttt{res}} = (\hat{\texttt{n}} \in \hat{\texttt{c}})\}$

- Un posible algoritmo para dicha función es el siguiente:

  iPertenece(in $\texttt{n} : \texttt{nat}$, in $\texttt{e} : \texttt{estr}$) $\rightarrow \texttt{res} : \texttt{bool}$

  $\texttt{res} \leftarrow (e.\mathit{lower} \leq n \leq e.\mathit{upper}) \wedge_L e.\mathit{elems}[n - e.\mathit{lower}]$

- El algoritmo recibe como parámetro la estructura de representación `estr` (en lugar del tipo representado `conjenrango`).

Representación: Algoritmo para Agregar

- Consideremos la siguiente operación declarada en la interfaz:

  Agregar(in `n` : `nat`, inout `c` : `conjenrango`)

  $\{\text{low}(\hat{c}) \leq \hat{n} \leq \text{up}(\hat{c}) \wedge c_0 = \hat{c}\}$

  $\{\hat{c} = \text{Ag}(\hat{n}, c_0)\}$

- Proponemos la siguiente:

```
iAgregar(in n : nat, inout e : estr)
    e.elems[n − e.lower] ← true
```

Representación: Algoritmo para Vacío

- En la interfaz declaramos la operación:

  Vacío(in $\mathtt{l} : \mathtt{nat}$, in $\mathtt{u} : \mathtt{nat}$) $\rightarrow \mathtt{res} : \mathtt{conjenrango}$

  $\{\hat{\mathtt{l}} \leq \hat{\mathtt{u}}\}$

  $\{\widehat{\mathtt{res}} = \emptyset(\hat{\mathtt{l}}, \hat{\mathtt{u}})\}$

- Un posible algoritmo para dicha operación es el siguiente:

  iVacío(in $\mathtt{l} : \mathtt{nat}$,in $\mathtt{u} : \mathtt{nat}$) $\rightarrow \mathtt{res} : \mathtt{estr}$

  $\mathtt{res} \leftarrow \langle \mathtt{l}, \mathtt{u}, \mathit{CrearArreglo}(\mathtt{u} - \mathtt{l} + 1, \mathit{false}) \rangle$

- Aquí recurrimos a la operación *CrearArreglo* del módulo ARRD.

Representación: Algoritmos para Lower y Upper

- Ahora consideramos las siguientes operaciones que habíamos declarado como:

  Lower(in c : `conjenrango`) $\rightarrow$ `res` : `nat`

  $\{\text{true}\}$

  $\{\widehat{\texttt{res}} = \text{low}(\hat{c})\}$

  Upper(in c : `conjenrango`) $\rightarrow$ `res` : `nat`

  $\{\text{true}\}$

  $\{\widehat{\texttt{res}} = \text{up}(\hat{c})\}$

- Proponemos los siguientes algoritmos

  iLower(in e : `estr`) $\rightarrow$ `res` : `nat`

  `res` $\leftarrow$ e.*lower*

  iUpper(in e : `estr`) $\rightarrow$ `res` : `nat`

  `res` $\leftarrow$ e.*upper*

Representación: Algoritmos para Cardinal y Borrar

- Ahora nos quedan las últimas dos operaciones, estos son posibles algoritmos:

```
iCardinal(in e : estr) → res : nat
    res ← 0
    for i ← 0 to tam(e.elems) − 1 do
        if e.elems[i] then res ← res + 1 fi
    end for

iBorrar(in n : nat, inout e : estr)
    if e.lower ≤ n ≤ e.upper then
        e.elems[n − e.lower] ← false
    fi
```

Servicios Usados

- En esta última sección del módulo de abstracción detallaremos todos los requisitos sobre las operaciones que se invocan de otros módulos y que permiten justificar las promesas en la interfaz propia.

Relación entre operaciones del TAD y del tipo diseñado

- No siempre hay una correspondencia 1 a 1 entre las operaciones del TAD y las del módulo de abstracción.
- Por ejemplo, el módulo PILA podría exportar una única función que desapile y devuelva el tope a la vez:

$$\text{Desapilar}(\text{inout } \texttt{p} : \texttt{pila}(\alpha)) \rightarrow \texttt{res} : \alpha$$
$$\{\neg \text{ vacía}(\hat{\texttt{p}}) \wedge (p_0 = \hat{\texttt{p}})\}$$
$$\{\hat{\texttt{p}} = \text{desapilar}(p_0) \wedge \widehat{\texttt{res}} = \text{tope}(p_0))\}$$

Punteros

&• : `tipo_dato` $\rightarrow$ `puntero(tipo_dato)`
Retorna la ubicación en memoria de la variable indicada.

*• : `puntero(tipo_dato)` $\rightarrow$ `tipo_dato`
Permite acceder al dato ubicado en la dirección de memoria.

NULL : `puntero(tipo_dato)`
Devuelve un puntero que no apunta a nada.

Usaremos $a \rightarrow b$ como azúcar sintáctico de $(*a).b$.