Introducción al diseño

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

Algoritmos y Estructuras de Datos 2
2do cuatrimestre de 2015

Diferencias entre especificación y diseño

- En la etapa de especificación:

Diferencias entre especificación y diseño

- En la etapa de especificación:
  - Nos ocupamos del '¿qué?'

Diferencias entre especificación y diseño

- En la etapa de especificación:
  - Nos ocupamos del '¿qué?'
  - Lo explicamos usando TADs

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

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:

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?'

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*

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

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

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

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

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

generadores

$\emptyset \; : \; \text{nat } l \times \text{nat } u \longrightarrow \text{conjran}$ $\qquad l \leq u$

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

otras operaciones

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

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

Fin TAD

Ejemplo: Conjunto en rango

TAD CONJUNTO EN RANGO

axiomas $\forall c$: conjran, $\forall l, u, n, n'$: 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

Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

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).

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).

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.

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.

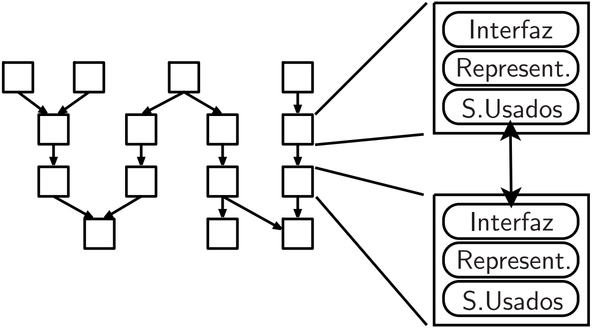
Partes de un módulo

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.

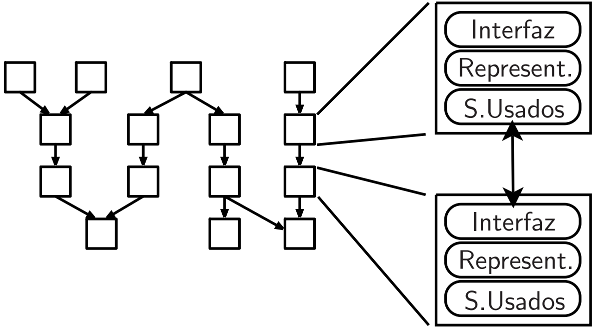
Interfaz y Servicios Usados

- En el siguiente diagrama cada cuadradito representa un módulo y las flechas indican la relación "usa a".



Interfaz y Servicios Usados

- En el siguiente diagrama cada cuadradito representa un módulo y las flechas indican la relación "usa a".



- Los servicios exportados en la interfaz de un módulo deben satisfacer los requisitos impuestos por las secciones de Servicios Usados de todos los módulos que lo utilizan.

Interfaz

En la sección Interfaz:

- se declaran todas las operaciones disponibles a los usuarios del módulo (y otros servicios como los iteradores).

Interfaz

En la sección Interfaz:

- se declaran todas las operaciones disponibles a los usuarios del módulo (y otros servicios como los iteradores).
- se detalla la complejidad temporal de cada una de las operaciones y posibles efectos secundarios y de ‘aliasing’ de cada una de ellas.

Interfaz

En la sección Interfaz:

- se declaran todas las operaciones disponibles a los usuarios del módulo (y otros servicios como los iteradores).
- se detalla la complejidad temporal de cada una de las operaciones y posibles efectos secundarios y de 'aliasing' de cada una de ellas.
- constan los parámetros formales y las dependencias con otros módulos.

Interfaz

En la sección Interfaz:

- se declaran todas las operaciones disponibles a los usuarios del módulo (y otros servicios como los iteradores).
- se detalla la complejidad temporal de cada una de las operaciones y posibles efectos secundarios y de 'aliasing' de cada una de ellas.
- constan los parámetros formales y las dependencias con otros módulos.

La sección Interfaz permite poner una capa de abstracción sobre nuestro módulo de manera de tener la posibilidad de cambiar la representación para el tipo que estamos diseñando sin tener que cambiar la forma en que los otros módulos acceden a las operaciones.

Interfaz: Encabezado

interfaz CONJENRANGO
usa: BOOL, NAT, . . .
se explica con: CONJUNTO EN RANGO
género: `conjenrango`
operaciones:
. . .

Interfaz: Operaciones

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

Interfaz: Operaciones

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

- Signatura

Interfaz: Operaciones

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

- Signatura
- Pre y postcondiciones

Interfaz: Operaciones

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

- Signatura
- Pre y postcondiciones
- Complejidad temporal

Interfaz: Operaciones

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

- Signatura
- Pre y postcondiciones
- Complejidad temporal
- Aliasing

Interfaz: Signatura

- Consideremos la siguiente operación del TAD:

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

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:

  $$\boxed{\text{Pertenece}(\text{in } \mathtt{n} : \mathtt{nat}, \text{ in } \mathtt{c} : \mathtt{conjenrango}) \to \mathtt{res} : \mathtt{bool}}$$

Interfaz: Signatura

- Consideremos la siguiente operación del TAD:

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

- Proponemos la siguiente signatura en la interfaz del diseño:

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

- 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.

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

$\{$true$\}$ [Precondición]

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

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 $\mathtt{n}$: $\mathtt{nat}$, in $\mathtt{c}$: $\mathtt{conjenrango}$) $\rightarrow$ $\mathtt{res}$: $\mathtt{bool}$

$\{$true$\}$ [Precondición]

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

- Las pre y postcondiciones se escriben usando las operaciones de TADs. Por eso aquí $\widehat{\mathtt{res}}$, $\hat{\mathtt{n}}$ y $\hat{\mathtt{c}}$ son las instancias de los TADs Bool, Nat y Conjunto en Rango que se corresponden con las instancias $\mathtt{res}$, $\mathtt{n}$ y $\mathtt{c}$ del mundo del diseño.

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 $\mathtt{n}$: $\mathtt{nat}$, in $\mathtt{c}$: $\mathtt{conjenrango}$) $\rightarrow$ $\mathtt{res}$: $\mathtt{bool}$
$\{$true$\}$ [Precondición]
$\{\widehat{\mathtt{res}} = (\hat{\mathtt{n}} \in \hat{\mathtt{c}})\}$ [Postcondición]

- Las pre y postcondiciones se escriben usando las operaciones de TADs. Por eso aquí $\widehat{\mathtt{res}}$, $\hat{\mathtt{n}}$ y $\hat{\mathtt{c}}$ son las instancias de los TADs Bool, Nat y Conjunto en Rango que se corresponden con las instancias $\mathtt{res}$, $\mathtt{n}$ y $\mathtt{c}$ 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

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

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

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

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

- Y para la operación del TAD

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

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

  $\text{Lower}(\text{in c} : \texttt{conjenrango}) \rightarrow \texttt{res} : \texttt{nat}$

  $\{\text{true}\}$

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

- Y para la operación del TAD

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

  daremos la siguiente declaración en la interfaz

  $\text{Upper}(\text{in c} : \texttt{conjenrango}) \rightarrow \texttt{res} : \texttt{nat}$

  $\{\text{true}\}$

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

Interfaz: Operaciones $\emptyset$ y Ag

- Para la operación del TAD

$$\emptyset \ : \ \textsf{nat}\ \ell \times \textsf{nat}\ u \ \longrightarrow \ \textsf{conjran} \qquad\qquad (\ell \leq u)$$

Interfaz: Operaciones $\emptyset$ y Ag

- Para la operación del TAD

$$\emptyset \;:\; \text{nat } \ell \times \text{nat } u \;\longrightarrow\; \text{conjran} \qquad (\ell \leq u)$$

la declararemos en la interfaz como

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

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

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

Interfaz: Operaciones $\emptyset$ y Ag

- Para la operación del TAD

  $\emptyset \; : \; \text{nat } \ell \times \text{nat } u \; \longrightarrow \; \text{conjran} \qquad\qquad (\ell \leq u)$

  la declararemos en la interfaz como

Vacío(in $\mathtt{l}$: $\mathtt{nat}$, in $\mathtt{u}$: $\mathtt{nat}$) $\rightarrow$ $\mathtt{res}$: $\mathtt{conjenrango}$
$\{\hat{\mathtt{l}} \leq \hat{\mathtt{u}}\}$
$\{\widehat{\mathtt{res}} = \emptyset(\hat{\mathtt{l}}, \hat{\mathtt{u}})\}$

- Y para la operación del TAD

  $\text{Ag} \; : \; \text{nat } n \times \text{conjran } c \; \longrightarrow \; \text{conjran}$

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

Interfaz: Operaciones $\emptyset$ y Ag

- Para la operación del TAD

  $$\emptyset \;:\; \mathrm{nat}\ \ell \times \mathrm{nat}\ u \longrightarrow \mathrm{conjran} \qquad (\ell \leq u)$$

  la declararemos en la interfaz como

Vacío(in $\mathtt{l}$: $\mathtt{nat}$, in $\mathtt{u}$: $\mathtt{nat}$) $\to$ $\mathtt{res}$: $\mathtt{conjenrango}$
$\{\hat{\mathtt{l}} \leq \hat{\mathtt{u}}\}$
$\{\widehat{\mathtt{res}} = \emptyset(\hat{\mathtt{l}}, \hat{\mathtt{u}})\}$

- Y para la operación del TAD

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

  la podemos declarar en la interfaz mediante

Agregar(in $\mathtt{n}$: $\mathtt{nat}$, inout $\mathtt{c}$: $\mathtt{conjenrango}$)
$\{\mathrm{low}(\hat{\mathtt{c}}) \leq \hat{\mathtt{n}} \leq \mathrm{up}(\hat{\mathtt{c}}) \wedge c_0 = \hat{\mathtt{c}}\}$
$\{\hat{\mathtt{c}} = \mathrm{Ag}(\hat{\mathtt{n}}, c_0)\}$

Interfaz: Operaciones $\#$ y $-$

- Para la operación del TAD

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

Interfaz: Operaciones # y −

- Para la operación del TAD

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

  la declararemos en la interfaz como

  $\text{Cardinal}(\text{in } \mathtt{c} : \mathtt{conjenrango}) \rightarrow \mathtt{res} : \mathtt{nat}$

  $\{\textit{true}\}$

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

Interfaz: Operaciones $\#$ y $-$

- Para la operación del TAD

  $\#$: conjran $\longrightarrow$ nat

  la declararemos en la interfaz como

Cardinal(in $\texttt{c}$: $\texttt{conjenrango}$) $\rightarrow$ $\texttt{res}$: $\texttt{nat}$
$\{\mathit{true}\}$
$\{\widehat{\texttt{res}} = \#(\hat{\texttt{c}})\}$

- Y por último para la operación

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

Interfaz: Operaciones # y −

- Para la operación del TAD

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

  la declararemos en la interfaz como

  > Cardinal(in $\mathtt{c} : \mathtt{conjenrango}) \to \mathtt{res} : \mathtt{nat}$
  >
  > $\{\mathit{true}\}$
  >
  > $\{\widehat{\mathtt{res}} = \#(\hat{\mathtt{c}})\}$

- 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 $\mathtt{n} : \mathtt{nat}$, inout $\mathtt{c} : \mathtt{conjenrango}$)
  >
  > $\{c_0 = \hat{\mathtt{c}}\}$
  >
  > $\{\hat{\mathtt{c}} = c_0 - \{\hat{\mathtt{n}}\}\}$

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.

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.

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.

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.

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.

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$(\alpha))$ $(\mathrm{tam}(a) =_{\mathrm{obs}} \mathrm{tam}(a') \wedge$
$(\forall n : \mathrm{nat})((n < \mathrm{tam}(a)) \implies a[n] =_{\mathrm{obs}} a'[n])))$

observadores básicos

$\mathrm{tam} \ : \ \mathrm{ad}(\alpha) \longrightarrow \mathrm{nat}$

$\bullet[\bullet] \ : \ \mathrm{ad}(\alpha)\ a \times \mathrm{nat}\ n \longrightarrow \alpha \qquad n < \mathrm{tam}(a)$

generadores

$\mathrm{crearArreglo} \ : \ \mathrm{nat} \times \alpha \longrightarrow \mathrm{ad}(\alpha)$

$\bullet[\bullet] \leftarrow \bullet \ : \ \mathrm{ad}(\alpha)\ a \times \mathrm{nat}\ n \times \alpha \longrightarrow \mathrm{ad}(\alpha) \qquad n < \mathrm{tam}(a)$

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

$\mathrm{tam}(\mathrm{crearArreglo}(n, v)) \equiv n$

$\mathrm{tam}(a\,[\,n\,] \leftarrow e) \equiv \mathrm{tam}(a)$

$(\mathrm{crearArreglo}(n, v))\,[\,m\,] \equiv v$

$(a\,[\,n\,] \leftarrow e)\,[\,m\,] \equiv$ **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:

$$
\begin{array}{l}
\texttt{conjenrango}\ \textbf{se representa con}\ \texttt{estr}\ \text{donde} \\[1em]
\texttt{estr}\ \text{es}\ \texttt{tupla}\langle \mathit{lower} : \texttt{nat} \times \\
\qquad\qquad\qquad\ \ \mathit{upper} : \texttt{nat} \times \\
\qquad\qquad\qquad\ \ \mathit{elems} : \texttt{arrd}(\texttt{bool})\rangle
\end{array}
$$

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

Representación: Estructura de representación

- Elegimos la siguiente representación para el tipo:

  $\texttt{conjenrango}$ **se representa con** $\texttt{estr}$ donde

  $$\begin{aligned} \texttt{estr} \text{ es } \texttt{tupla}\langle & \textit{lower} : \texttt{nat} \times \\ & \textit{upper} : \texttt{nat} \times \\ & \textit{elems} : \texttt{arrd}(\texttt{bool})\rangle \end{aligned}$$

  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`.

Representación: Estructura de representación

- Elegimos la siguiente representación para el tipo:

  `conjenrango` **se representa con** `estr` donde

  $$\texttt{estr}\ \text{es}\ \texttt{tupla}\langle \textit{lower} : \texttt{nat} \times \\ \textit{upper} : \texttt{nat} \times \\ \textit{elems} : \texttt{arrd}(\texttt{bool})\rangle$$

  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.

Relacionando

$$\begin{tikzcd}
\texttt{conjuntoDeNats} \arrow[r, "\text{se representa con}"] \arrow[d, "\text{se explica con (\^{})}"'] & \texttt{estr} = \texttt{arregloDeNats} \arrow[dl, "\mathrm{Abs}"] \arrow[d, "\hat{\ }"'] \\
\mathsf{conj(nat)} & \widehat{\texttt{estr}} = \mathsf{ad(nat)}
\end{tikzcd}$$

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 $\text{Rep}(e)$ es verdadera serán las representaciones 'admisibles', es decir, aquellas que bajo nuestra interpretación vamos a asignarle una determinada instancia del TAD que estamos diseñando.

Representación: Invariante de representación (Rep)

- El invariante de representación tiene la siguiente signatura:

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

  Aquellas instancias e del tipo $\widehat{\texttt{estr}}$ tales que $\mathrm{Rep}(e)$ 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 función que se le aplique al módulo.

Representación: Invariante de representación (Rep)

- Como invariante de representación proponemos:

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

$$(\forall e : \widehat{\texttt{estr}})(\text{Rep}(e) =$$

$$(e.lower \leq e.upper) \land$$

$$(tam(e.elems) \equiv e.upper - e.lower + 1))$$

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.

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

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

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

$$\boxed{\begin{array}{l} \text{Abs} : \widehat{\texttt{estr}}\ e \to \text{conjran} \\ \text{Abs}(e) = c \ / \ ((\text{low}(c) = e.\textit{lower}) \wedge (\text{up}(c) = e.\textit{upper}) \wedge_L \\ \qquad (\forall n : \text{nat})(n \in c \Leftrightarrow ((\text{low}(c) \leq n \leq \textit{up}(c)) \\ \qquad \wedge_L e.\textit{elems}[n - \textit{low}(c)]))) \end{array}}$$

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

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

- Para el caso del TAD Conjunto en Rango proponemos

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

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

- 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.

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 $\mathtt{n}$: $\mathtt{nat}$, in $\mathtt{c}$: $\mathtt{conjenrango}$) $\rightarrow$ $\mathtt{res}$: $\mathtt{bool}$

  $\{\text{true}\}$

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

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 $\texttt{e}$: $\texttt{estr}$) $\rightarrow$ $\texttt{res}$: $\texttt{bool}$

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

Representación: Algoritmo para Pertenece

- En la interfaz habíamos declarado la operación:

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

  $\{\text{true}\}$

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

- Un posible algoritmo para dicha función es el siguiente:

$$\boxed{\begin{array}{l} \text{iPertenece(in } \mathtt{n} : \mathtt{nat}\text{, in } \mathtt{e} : \mathtt{estr}) \rightarrow \mathtt{res} : \mathtt{bool} \\ \quad \mathtt{res} \leftarrow (\mathit{e.lower} \leq n \leq \mathit{e.upper}) \wedge_L \mathit{e.elems}[n - \mathit{e.lower}] \end{array}}$$

- 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 $\mathtt{n}$: $\mathtt{nat}$, inout $\mathtt{c}$: $\mathtt{conjenrango}$)

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

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

Representación: Algoritmo para Agregar

- Consideremos la siguiente operación declarada en la interfaz:

  Agregar(in $\mathtt{n}$: $\mathtt{nat}$, inout $\mathtt{c}$: $\mathtt{conjenrango}$)

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

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

- Proponemos la siguiente:

iAgregar(in $\mathtt{n}$: $\mathtt{nat}$, inout $\mathtt{e}$: $\mathtt{estr}$)
$e.elems[n - e.lower] \leftarrow true$

Representación: Algoritmo para Vacío

- En la interfaz declaramos la operación:

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

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

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

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

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 $\texttt{c}$: $\texttt{conjenrango}$) $\rightarrow$ $\texttt{res}$: $\texttt{nat}$

  $\{\text{true}\}$

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

  Upper(in $\texttt{c}$: $\texttt{conjenrango}$) $\rightarrow$ $\texttt{res}$: $\texttt{nat}$

  $\{\text{true}\}$

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

Representación: Algoritmos para Lower y Upper

- Ahora consideramos las siguientes operaciones que habíamos declarado como:

  Lower(in $\mathtt{c}$: $\mathtt{conjenrango}$) $\rightarrow$ $\mathtt{res}$: $\mathtt{nat}$

  $\{\text{true}\}$

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

  Upper(in $\mathtt{c}$: $\mathtt{conjenrango}$) $\rightarrow$ $\mathtt{res}$: $\mathtt{nat}$

  $\{\text{true}\}$

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

- Proponemos los siguientes algoritmos

iLower(in $\mathtt{e}$: $\mathtt{estr}$) $\rightarrow$ $\mathtt{res}$: $\mathtt{nat}$

  $\mathtt{res} \leftarrow \mathtt{e}.\textit{lower}$

iUpper(in $\mathtt{e}$: $\mathtt{estr}$) $\rightarrow$ $\mathtt{res}$: $\mathtt{nat}$

  $\mathtt{res} \leftarrow \mathtt{e}.\textit{upper}$

Representación: Algoritmos para Cardinal y Borrar

Representación: Algoritmos para Cardinal y Borrar

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

iCardinal(in $\mathtt{e}$: $\mathtt{estr}$) $\rightarrow$ $\mathtt{res}$: $\mathtt{nat}$

  $\mathtt{res} \leftarrow 0$

  for $\mathtt{i} \leftarrow 0$ **to** $\mathit{tam}(\mathtt{e}.\mathit{elems}) - 1$ **do**

    if $\mathtt{e}.\mathit{elems}[\mathtt{i}]$ **then** $\mathtt{res} \leftarrow \mathtt{res} + 1$ **fi**

  end for

iBorrar(in $\mathtt{n}$: $\mathtt{nat}$, inout $\mathtt{e}$: $\mathtt{estr}$)

  if $\mathtt{e}.\mathit{lower} \leq \mathtt{n} \leq \mathtt{e}.\mathit{upper}$ **then**

    $\mathtt{e}.\mathit{elems}[\mathtt{n} - \mathtt{e}.\mathit{lower}] \leftarrow \mathit{false}$

  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.

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:

Desapilar(inout $\mathtt{p} : \mathtt{pila}(\alpha)) \rightarrow \mathtt{res} : \alpha$

$\{\neg\,\text{vacía}(\hat{\mathtt{p}}) \wedge (p_0 = \hat{\mathtt{p}})\}$

$\{\hat{\mathtt{p}} = \text{desapilar}(p_0) \wedge \widehat{\mathtt{res}} = \text{tope}(p_0))\}$

Aliasing

- Supongamos que el módulo CONJENRANGO permite hacer copias de sus instancias mediante la operación:

$$\text{Replicar}(\text{in } \mathtt{c} : \mathtt{conjenrango}) \rightarrow \mathtt{res} : \mathtt{conjenrango}$$

$$\{\text{true}\}$$

$$\{\widehat{\mathtt{res}} = \hat{\mathtt{c}}\}$$

Aliasing

- Supongamos que el módulo CONJENRANGO permite hacer copias de sus instancias mediante la operación:

Replicar(in $\mathtt{c}$: $\mathtt{conjenrango}$) $\rightarrow$ $\mathtt{res}$: $\mathtt{conjenrango}$
$\{\mathrm{true}\}$
$\{\widehat{\mathtt{res}} = \hat{\mathtt{c}}\}$

  para la cual se propuso el siguiente algoritmo:

iReplicar(in $\mathtt{e}$: $\mathtt{estr}$) $\rightarrow$ $\mathtt{res}$: $\mathtt{estr}$
$\quad \mathtt{res} \leftarrow \mathtt{e}$

Aliasing

- Supongamos que el módulo CONJENRANGO permite hacer copias de sus instancias mediante la operación:

  Replicar(in $\texttt{c}$: $\texttt{conjenrango}$) $\rightarrow$ $\texttt{res}$: $\texttt{conjenrango}$

  $\{\mathsf{true}\}$

  $\{\widehat{\texttt{res}} = \hat{\texttt{c}}\}$

  para la cual se propuso el siguiente algoritmo:

  iReplicar(in $\texttt{e}$: $\texttt{estr}$) $\rightarrow$ $\texttt{res}$: $\texttt{estr}$

  $\quad\texttt{res} \leftarrow \texttt{e}$

- Como no es una asignación entre tipos básicos, aquí se produce aliasing (dos nombres para un mismo objeto).

Aliasing

- Supongamos que el módulo CONJENRANGO permite hacer copias de sus instancias mediante la operación:

  Replicar(in $\mathtt{c}$: $\mathtt{conjenrango}$) $\rightarrow$ $\mathtt{res}$: $\mathtt{conjenrango}$

  $\{\text{true}\}$

  $\{\widehat{\mathtt{res}} = \hat{\mathtt{c}}\}$

  para la cual se propuso el siguiente algoritmo:

  iReplicar(in $\mathtt{e}$: $\mathtt{estr}$) $\rightarrow$ $\mathtt{res}$: $\mathtt{estr}$

  $\quad\mathtt{res} \leftarrow \mathtt{e}$

- Como no es una asignación entre tipos básicos, aquí se produce aliasing (dos nombres para un mismo objeto).
- Este aspecto debe constar explícitamente en la declaración de la función interfaz.

Punteros

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

*• : `puntero(tipo_dato)` → `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 $(*a).b$.

Al pizarrón

TAD $\text{Conjunto}(\alpha)$

observadores básicos

$\cdot \in \cdot \;:\; \alpha \times \text{conj}(\alpha) \longrightarrow \text{bool}$

generadores

$\emptyset \;:\; \longrightarrow \text{conj}(\alpha)$

$\text{Ag} \;:\; \alpha \times \text{conj}(\alpha) \longrightarrow \text{conj}(\alpha)$

axiomas

$a \in \emptyset \equiv \text{false}$

$a \in \text{Ag}(b, c) \equiv (a = b) \vee (a \in c)$

Fin TAD

Al pizarrón

TAD $\textsc{Conjunto}(\alpha)$

observadores básicos

$\cdot \in \cdot \; : \; \alpha \times \text{conj}(\alpha) \longrightarrow \text{bool}$

generadores

$\emptyset \; : \; \longrightarrow \text{conj}(\alpha)$

$\text{Ag} \; : \; \alpha \times \text{conj}(\alpha) \longrightarrow \text{conj}(\alpha)$

otras operaciones

$\#\text{MenoresQue} \; : \; \alpha \times \text{conj}(\alpha) \longrightarrow \text{nat}$

axiomas

$$
\begin{array}{lcl}
a \in \emptyset & \equiv & \text{false} \\
a \in \text{Ag}(b,\ c) & \equiv & (a = b) \vee (a \in c) \\
\#\text{MenoresQue}(e, \emptyset) & \equiv & 0 \\
\#\text{MenoresQue}(e, Ag(b, c)) & \equiv & \textbf{if } e < b \wedge \neg(b \in c) \textbf{ then } 1 \\
 & & \textbf{else } 0 \textbf{ fi} + \\
 & & \#\text{MenoresQue}(e, c)
\end{array}
$$

Fin TAD