

# Introducción al diseño de TADs 2

Ariel Bendersky<sup>1</sup>, Nicolás D'Ippolito<sup>1</sup>

<sup>1</sup>Departamento de Computación, FCEyN,  
Universidad de Buenos Aires, Buenos Aires, Argentina

Algoritmos y Estructuras de Datos II  
Segundo cuatrimestre de 2019

## (2) Dónde estamos

- Vimos
  - por qué es necesaria la etapa de diseño,
  - qué cambios introduce, y
  - cuál es el lenguaje que usamos.
- Es decir:
  - La diferencia de mundos.
  - Cómo los requerimientos de eficiencia deciden la implementación.
  - La idea de diseño top-down.
  - El cambio de paradigma.
    - Aliasing
    - Sombrerito.
- Veremos hoy:
  - Cómo escribir un módulo.
  - Qué cosas debemos considerar.
  - Cómo verificar su relación con el TAD.

### (3) Ocultando información

- ¿Por qué tanto énfasis en la interfaz?
- ¿No es más fácil dar el código y listo?
- Primera piedra: *Information hiding*, David Parnas, “On the Criteria to Be Used in Decomposing Systems Into Modules” (Communications of the ACM, Diciembre de 1972).
- Tres definiciones...

## (4) Ocultando información (cont.)

- **Abstracción:** “Abstraction is a process whereby we identify the important aspects of a phenomenon and ignore its details.” [Ghezzi et al, 1991]
- **Information hiding:**
  - “The [...] decomposition was made using 'information hiding' [...] as a criterion. [...] Every module [...] is characterized by its knowledge of a design decision which it hides from all others. Its interface or definition was chosen to reveal as little as possible about its inner workings.” [Parnas, 1972b]
  - “[...] the purpose of hiding is to make inaccessible certain details that should not affect other parts of a system.” [Ross et al, 1975]
- **Encapsulamiento:** “[...] A consumer has full visibility to the procedures offered by an object, and no visibility to its data. From a consumer's point of view, an object is a seamless capsule that offers a number of services, with no visibility as to how these services are implemented [...] The technical term for this is encapsulation.” [Cox, 1986]

## (5) Ocultando información (cont.)



Ventajas del ocultamiento, la abstracción y el encapsulamiento:

- La implementación se puede cambiar y mejorar sin afectar su uso.
  - Ayuda a modularizar.
  - Facilita la comprensión.
  - Favorece el reuso.
  - Los módulos son más fáciles de entender.
  - Y de programar.
  - El sistema es más resistente a los cambios.
- Aprender a elegir buenas descomposiciones no es fácil. Ese aprendizaje comienza ahora y continúa en Ingeniería del Software I.


## (6) Information hiding, ¿hasta dónde?

- Dijimos que nos íbamos a basar en ocultar la información.
- ¿De quién ocultamos las cosas?
- No nos olvidemos de que si bien nos *hacemos* los misteriosos, las promesas hay que cumplirlas.
- La interfaz es una promesa.

## (7) Manteniendo nuestra palabra

- Es hora de documentar nuestra estructura.
- Ejemplo: conjunto semi rápido de naturales. Los números del 1 al 100 deben manejarse en  $O(1)$  porque se usan mucho. El resto, en  $O(n)$ . Rápidamente debo conocer la cardinalidad.
- Propuesta:
  - Un arreglo de 100 posiciones booleanas.
  - Una secuencia.
  - Un nat para la cardinalidad.
- En nuestro lenguaje de diseño, se expresa así:

conj\_semi\_rápido\_nat **se representa con**  
tupla <rápido: arreglo [0..100] de bool  $\times$  resto: secu(nat)  
 $\times$  cant: nat>

•  
 ¿Y dónde aprendo el lenguaje? → En el apunte de diseño.

## (8) (¿Cómo elegir la representación adecuada?)

- ¿Se acuerdan?

 Contexto de uso y requerimientos de eficiencia.

- Todo un tema que iremos viendo en la segunda parte de la materia.
- Vamos a suponer que eso ya lo tenemos resuelto, para poder analizar otros aspectos.



## (9) Estructura de representación

- Identifiquemos las partes

conj\_semi\_rápido\_nat **src** estr  
**donde** estr **es** tupla <rápido: arreglo [0..100] de bool ×  
resto: secu(nat) × cant: nat>

- 
- conj\_semi\_rápido\_nat, la tupla, la secu, etc. “son de bits”.
- **src** es una abreviatura de “**se representa con**”.
- estr es una *macro* que se expande en la tupla.
- conj\_semi\_rápido\_nat es el **género representado** y estr (su expansión) es el **genero de representación**.

## (10) ¿Cualquier instancia es válida?

- $\langle [0...0], \langle \rangle, 8254 \rangle$  es un `conj_semi_rápido_nat` válido?



¿Para qué nos serviría poder separar con facilidad instancias válidas e inválidas?

- Como una forma de documentar la estructura.
  - Como condición necesaria para establecer una relación con la abstracción (ver más adelante).
  - Para agregar a las postcondiciones, como una forma de garantizar que nuestros algoritmos no rompen la estructura.
  - Para agregar a las precondiciones, como una “garantía” con la que cuentan nuestros algoritmos.
  - Como una guía a la hora de escribir los algoritmos.
  - Si pudiésemos programar el chequeo, como una forma de detectar instancias corruptas.
- El **invariante de representación**.

## (11) Invariante de representación

⚠ Es una función booleana con dominio en el género de representación que da *true* cuando recibe una instancia válida.

- ¿Podría el dominio ser el género representado? ¿Por qué?
- En realidad, si nos ponemos finos, el dominio es la **versión abstracta** del género de representación.

### Ejemplo

Si representamos  $T_1$  sobre  $T_2$

Rep:  $\widehat{T}_2 \rightarrow \text{bool}$

$(\forall t : \widehat{T}_2) \text{Rep}(t) \equiv \dots$  condiciones que garanticen que  $t$  representa una instancia válida de  $T_1 \dots$

## (12) Invariante de representación (cont.)

- Recordemos nuestro ejemplo:

`conj_semi_rápido_nat` **se representa con**  
tupla  $\langle \text{rápido: arreglo } [0..100] \text{ de bool} \times \text{resto: secu(nat)}$   
 $\times \text{cant: nat} \rangle$

- 
- ¿Qué debería decir el invariante?
  - 1 Que *resto* sólo tiene números mayores a 100, si tiene alguno.
  - 2 Que *resto* no tiene números repetidos.
  - 3 Que *cant* tiene la longitud de *resto* más la cantidad de celdas de *rápido* que están en *true*.
- Rep:  $\widehat{estr} \rightarrow \text{bool}$
- $(\forall e : \widehat{estr}) \text{Rep}(e) \equiv$ 
  - 1  $(\forall n : \text{nat}) (\text{esta?}(n, e.\text{resto}) \Rightarrow n > 100) \wedge$
  - 2  $(\forall n : \text{nat}) (\text{cant\_apariciones}(n, e.\text{resto}) \leq 1) \wedge$
  - 3  $e.\text{cant} = \text{long}(e.\text{resto}) + \text{contar\_trues}(e.\text{rápido})$

⚠ ¡El invariante cambia la vida!

## (14) ¿Cómo se “lee” nuestra estructura?

- ¿Cómo hay que entender a nuestra estr para pensarla como un conj\_semi\_rápido\_nat?
- Para responder a esa pregunta vamos a definir una **función de abstracción**:
- Abs:  $\widehat{T}_2 \text{ e} \rightarrow \widehat{T}_1 (\text{Rep}(\text{e}))$
- Notar la restricción.
- Toma una instancia (abstracta) de la estructura de representación y devuelve una instancia (también abstracta) del genero representado.
- ¿Por qué toma géneros abstractos? Porque en el mundo abstracto es donde (mejor) sabemos razonar sobre las cosas.

### En nuestro ejemplo

Abs:  $\widehat{estr} \text{ e} \rightarrow \widehat{conj\_semi\_rapido\_nat} (\text{Rep}(\text{e}))$

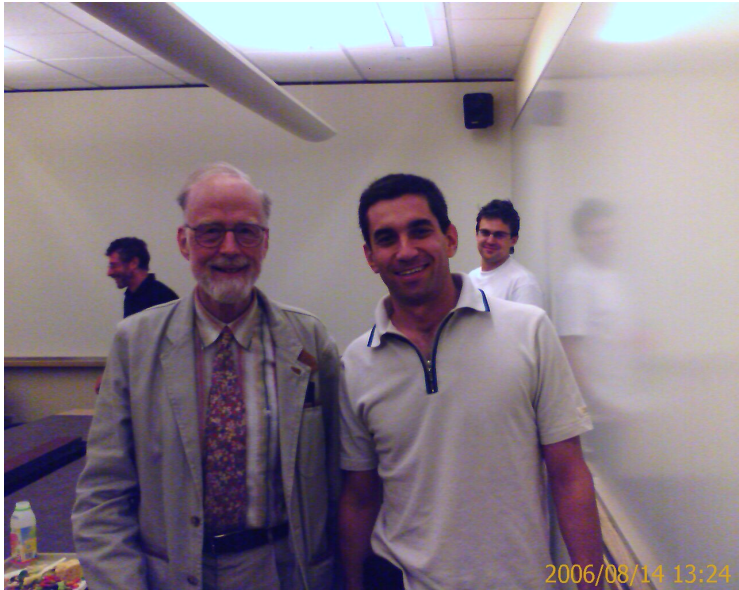
Abs(e)  $\equiv$  c /

$(\forall n: \text{nat}) (n \in c \iff ((n < 100 \wedge \text{e.rápido}[n]) \vee$   
 $(n \geq 100 \wedge \text{está?}(n, \text{e.resto})))$

## (15) Algunas notas sobre Abs.

- Hay otra forma de escribir Abs (sobre los generadores del tipo de representación en lugar de sobre los observadores del tipo representado), pero eso lo van a ver en la práctica.
- ¿Es total o parcial? Una vez restringida ( $\text{Rep}(e)$ ), deber ser total.
- ¿Debe ser sobreyectiva? Sí.
- ¿Debe ser inyectiva? No, pero puede serlo.
- Debe ser un homomorfismo respecto de la signature del TAD:
  - Para toda operación  $o$   $\text{Abs}(o(p_1, \dots, p_n)) \equiv o(\text{Abs}(p_1), \dots, \text{Abs}(p_n))$
- Abs y Rep se las debemos a [Hoare, 1972].

## (16) Identifique al prócer...





## (17) Un poco más de invariante

- Analicemos  $Ag()$ :

### Interfaz

$Ag(\text{inout } C: \text{conj\_semi\_rápido\_nat}, \text{in } e: \text{nat})$

$\{\hat{C} \equiv C_0 \wedge \hat{e} \notin C\}$

$\{\hat{C} \equiv \text{Agregar}(C_0, \hat{e})\}$



### Implementación

$iAg(\text{inout } C: \text{estr}, \text{in } e: \text{nat})$

$\{\text{Rep}(\hat{C}) \wedge_L \text{Abs}(\hat{C}) \equiv C_0 \wedge \hat{e} \notin \text{Abs}(\hat{C})\}$

$C.\text{cant}++$

if  $(e < 100)$  then

.  $C.\text{rápido}[e] := \text{true}$

else

.  $\text{ag\_en\_secu}(C, e)$

fi

$\{\text{Rep}(\hat{C}) \wedge_L \text{Abs}(\hat{C}) \equiv \text{Agregar}(C_0, \hat{e})\}$

$\text{ag\_en\_secu}(\text{inout } E: \text{estr}, \text{in } e: \text{nat})$

$\{\hat{E} \equiv E_0 \wedge \hat{e} \geq 100\}$

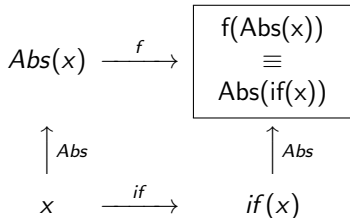
// **Notar: acá no vale**  
 $\text{Rep}(E)$

$\text{InsertarAlFinal}(E.\text{resto}, e)$

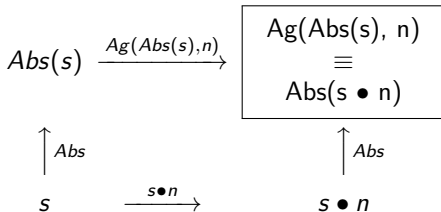
$\{\hat{E}.\text{resto} \equiv E_0.\text{resto} \bullet \hat{e}\}$

## (18) Probando corrección

Para toda operación  $f$  que implementa una operación del TAD y toda  $x$  instancia del género de representación, debemos ver que el siguiente diagrama conmuta:



Ejemplo para conjunto sobre secuencia ( $\forall s : secu, \forall n : nat$ )



## (19) Repaso

Vimos...

- La diferencia de mundos.
- Cómo los requerimientos de eficiencia deciden la implementación.
- La idea de diseño top-down.
- El cambio de paradigma.
  - Aliasing
  - Sombrero.
- Encapsulamiento.
- Abstracción.
- Ocultamiento de información.
- Relación entre el tipo representado y el de representación.
  - Invariante.
  - Función de abstracción.
- “Eso de la *i* que apareció por ahí”.

## (20) En las próximas clases

- Elección de estructuras.
- Cómo se propagan los contextos de uso y requerimientos de eficiencia.
- Cómo se escribe el código.
- Documentación.
- Ejemplos más complicados.

## (21) Bibliografía

- “Abstraction, Encapsulation, and Information Hiding”. By Edward V. Berard. The Object Agency.  
<http://www.itmweb.com/essay550.htm>
- [Parnas, 1972b] D.L. Parnas, “On the Criteria To Be Used in Decomposing Systems Into Modules,” Communications of the ACM, Vol. 5, No. 12, December 1972, pp. 1053-1058.
- [Ghezzi et al, 1991] C. Ghezzi, M. Jazayeri, and D. Mandrioli, Fundamentals of Software Engineering, Prentice-Hall, Englewood Cliffs, New Jersey, 1991.
- [Ross et al, 1975] D.T. Ross, J.B. Goodenough, and C.A. Irvine, “Software Engineering: Process, Principles, and Goals,” IEEE Computer, Vol. 8, No. 5, May 1975, pp. 17 - 27.
- [Cox, 1986] B.J. Cox, Object Oriented Programming: An Evolutionary Approach, Addison-Wesley, Reading, Massachusetts, 1986.
- [Hoare, 1972] C.A.R. Hoare. “Proof of correctness of Data Representation”. Acta Informatica 1(1), 1972.