Especificación 2

Flavia Bonomo (Sobre base de Fernando Schapachnik)[1]

[1]Departamento de Computación, FCEyN,
Universidad de Buenos Aires, Buenos Aires, Argentina

Algoritmos y Estructuras de Datos II, septiembre de 2009

(2) En el capítulo anterior...

- Vimos:
  - La necesidad de especificar como un paso previo a la resolución de problemas.
  - La conveniencia de hacerlo de manera formal.
  - Una introducción a los TADs.
  - El uso de cuantificadores.
  - Algunos tipos básicos: $\textsc{Bool}$, $\textsc{Nat}$.
- Hoy
  - Vamos a ver más tipos básicos, pero que tienen algunas particularidades.
  - Vamos a ver en detalle las partes de un TAD.
  - Vamos a hablar sobre especificación y modelado.
  - Vamos a ver un par de conceptos importantes.

(3) Pero primero...

- La especificación con TADs no es la única forma de hacerlo.
- Métodos informales.
  - Tienen menor costo inicial, pero no permiten encontrar errores e inconsistencias en la especificación, ni la aplicación de técnicas automáticas para la validación, etc.
  - El costo de reparar un defecto aumenta con el tiempo entre la introducción y el descubrimiento.
- Métodos formales de especificación, de variado tipo y color.
  - Algebraicos: el sistema se especifica mediante la definición de un álgebra con sus términos, operaciones, etc. Los TADs entran en esta categoría.
  - Operacionales: la especificación se realiza en un lenguaje imperativo de alto nivel.
  - Basados en estados: el sistema se modela como un conjunto de estados posibles y las relaciones entre ellos.

(4) Paréntesis lógico

- Lógica trivaluada: *true*, *false*, $\perp$.
- Versión $_L$ de los conectivos lógicos.
- Cuantificadores: (CUANT var: género) P(var)
- Variables ligadas:
  $((\forall x)\ P(x)) \wedge ((\exists y)\ Q(y)) \equiv ((\forall x)\ P(x)) \wedge ((\exists x)\ Q(x))$
- $(\exists x : nat)\ P(x) \approx P(0) \vee P(1) \vee \dots$
- $(\forall x)\ P(x) \equiv \neg((\exists x)\ \neg P(x))$ (ie, "todos los x satisfacen P" $\equiv$ "no hay ningún x que no cumpla P")
- "Expandiendo" el existencial y aplicando De Morgan:
  $\approx P(0) \wedge P(1) \wedge \dots$

⚠ Las expansiones son una "forma de decirlo", pero rigurosamente no es así: pensar en indefiniciones e infinitud.

(5) Paréntesis lógico (cont.)

- En general los vamos a usar con rangos. Eg, $(\forall x : nat)\ (1 \leq x) \Rightarrow_{\mathrm{L}} x/x = 1$
- Es decir, vamos a escribir: $(\forall x : \text{género})\ (R(x) \Rightarrow_{\mathrm{L}} P(x))$, donde R nos dice cuáles son los x sobre los que nos interesa el predicado P.
- ¿Qué pasa con el existencial? $(\exists x : \text{género})\ (R(x) \wedge_{\mathrm{L}} P(x))$.

⚠ Notar: si el rango es vacío ($R(x)$ no vale para ningún x) el $\forall$ da verdadero. En el caso de un $\exists$, da falso (pensar en las expansiones de la transparencia anterior).

- Nos vamos a permitir las siguientes macros: $(\forall x : nat, R(x))\ P(x) \equiv (\forall x : nat | R(x))\ P(x) \equiv (\forall x : nat)\ (R(x) \Rightarrow_{\mathrm{L}} P(x))$.
- Para el existencial: $(\exists x : nat, R(x))\ P(x) \equiv (\exists x : nat | R(x))\ P(x) \equiv (\exists x : nat)\ (R(x) \wedge_{\mathrm{L}} P(x))$.

TAD BOOL

géneros	bool	
exporta	bool, generadores, $\neg$, $\vee$, $\wedge$, $\Rightarrow$	

generadores

true	:		$\longrightarrow$ bool
false	:		$\longrightarrow$ bool

otras operaciones

$\neg\bullet$	:	bool	$\longrightarrow$ bool
$\bullet \vee \bullet$	:	bool $\times$ bool	$\longrightarrow$ bool
$\bullet \wedge \bullet$	:	bool $\times$ bool	$\longrightarrow$ bool
$\bullet \Rightarrow \bullet$	:	bool $\times$ bool	$\longrightarrow$ bool

Fin TAD

axiomas

$\neg true \equiv false$

$\neg false \equiv true$

$(\forall\ x : \text{bool})\ \ true \lor x \equiv true$

$(\forall\ x : \text{bool})\ \ false \lor x \equiv x$

$(\forall\ x : \text{bool})\ \ true \land x \equiv x$

$(\forall\ x : \text{bool})\ \ false \land x \equiv false$

$(\forall\ x, y : \text{bool})\ \ x \Rightarrow y \equiv \neg x \lor y$

(8) Secciones de un TAD

- **Géneros**. Los géneros (en general va a haber sólo uno, pero podrían ser más) son el nombre que recibe el conjunto de valores del tipo. Pensar en el monoide conmutativo $(\mathbb{N}, +)$ y en el conjunto de los números naturales $\mathbb{N}$.
- **Usa**. Inclusión de los genéros y operaciones exportadas de los tipos mencionados allí.
- **Exporta**. Qué operaciones y géneros se dejan a disposición de los usuarios del tipo.
- **Generadores.** Son operaciones que permiten construir valores del tipo. Un conjunto de generadores está bien armado si una combinación de ellos permite construir cualquier instancia posible del tipo.
- **Observadores**. Son aquellas operaciones que nos permiten, utilizadas en conjunto, diferenciar instancias del tipo.
- **Axiomas**. Son las reglas que nos explican el comportamiento de las funciones.

(9) TAD Conjunto(α)

TAD Conjunto(α)

parámetros formales

géneros α

operaciones

No se requiere ninguna en particular.

géneros conj(α)

observadores básicos

$\bullet \in \bullet$	: $\alpha \times$ conj(α)	$\longrightarrow$	bool	

generadores

$\emptyset$	:	$\longrightarrow$	conj(α)	
Ag	: $\alpha \times$ conj(α)	$\longrightarrow$	conj(α)	

otras operaciones

$\emptyset$?	: conj(α)	$\longrightarrow$	bool	
$\#$	: conj(α)	$\longrightarrow$	nat	
$\bullet$- $\{ \bullet \}$	: conj(α) $\times\ \alpha$	$\longrightarrow$	conj(α)	
dameUno	: conj(α) *c*	$\longrightarrow$	α	$\neg\emptyset?(c)$
sinUno	: conj(α) *c*	$\longrightarrow$	conj(α)	$\neg\emptyset?(c)$

axiomas $(\forall\ c, d : \text{conj}(\alpha))$, $(\forall\ a, b : \alpha)$

$$
\begin{array}{lcl}
a \in \emptyset & \equiv & \textit{false} \\
a \in \text{Ag}(b,\ c) & \equiv & (a \equiv b) \lor (a \in c) \\
\emptyset?(\emptyset) & \equiv & \textit{true} \\
\emptyset?(\text{Ag}(b,\ c)) & \equiv & \textit{false} \\
\#(\emptyset) & \equiv & 0 \\
\#(\text{Ag}(a,\ c)) & \equiv & 1 + \#(c \text{ - } \{\ a\ \}\ ) \\
\emptyset \text{ - } \{\ a\ \} & \equiv & \emptyset \\
\text{Ag}(a,\ c) \text{ - } \{\ b\ \} & \equiv & \textbf{if } a \equiv b \textbf{ then} \\
 & & \quad c \text{ - } \{\ b\ \} \\
 & & \textbf{else} \\
 & & \quad \text{Ag}(a,\ c \text{ - } \{\ b\ \}) \\
 & & \textbf{fi} \\
\text{dameUno}(c) \in c & & \\
\text{sinUno}(c) & \equiv & c \text{ - } \{\ \text{dameUno}(c)\ \}
\end{array}
$$

(11) Axiomatización

⚠ Las operaciones son funciones, así que debemos evitar inconsistencias. Para lograr esto, evitaremos la sobre especificación.

⚠ No hay pattern matching.

- prim_excepto_3(3 • S) $\equiv$ 21
- prim_excepto_3(a • S) $\equiv$ a
- Es una sobre especificación, que además produce una inconsistencia. La forma correcta es con un if.

⚠ Tampoco podemos especificar sobre los casos restringidos, ya que están fuera del dominio.

⚠ No debemos subespecificar.

- (Excepto que eso sea lo que estemos buscando, pero eso se da sólo en situaciones muy particulares).

(12) Axiomatización (cont.)

- Una regla práctica consiste en axiomatizar los observadores básicos sobre todos los generadores no restringidos, para asegurar que cubrimos todo el dominio.
- No siempre es necesario: doble_de_la_long(S) $\equiv$ long(S).2
- En general, aunque podría haber excepciones, el resto de las operaciones deberían poder escribirse en base los observadores básicos.

(13) Axiomatización (cont.)

⚠ Desde afuera de un tipo, sólo podemos usar los observadores:

- En el caso del restaurant, imaginemos la función platos_con_precio_X que toma un conjunto de platos y devuelve el subconjunto de los que valen X.
- platos_con_precio_X(r, $\emptyset$, x) $\equiv$ $\emptyset$
- platos_con_precio_X(r, Ag(p, c), x) $\equiv$ **if** precio(p, r)=x **then** Ag(p, platos_con_precio_X(r, c, x)) **else** platos_con_precio_X(r, c, x) **fi**

⚠ Es incorrecto (está haciendo pattern matching y viola el encapsulamiento). Desde el TAD restaurant tenemos que manipular al conjunto a través de sus observadores:

- platos_con_precio_X(r, c, x) $\equiv$ **if** $\emptyset$?(c) **then** $\emptyset$ **else** **if** precio(DameUno(c), r)=x **then** Ag(DameUno(c), platos_con_precio_X(r, SinUno(c), x)) **else** platos_con_precio_X(r, SinUno(c), x) **fi** **fi**

(14) Algunas libertades

- Se **usa** todo lo que aparece mencionado (ie, podemos no escribir la cláusula usa).
- El **exporta** tiene un valor por omisión: los géneros, los observadores básicos y generadores.
- Además, cuando se exporta algo más, alcanza con aclarar qué es lo extra que se exporta (lo demás se da por mencionado).
- En la axiomatización:
  - Todas las variables libres se suponen universalmente cuantificadas.
  - Sólo cuantificamos explícitamente cuando podría generarse confusión (por ejemplo, el caso del mínimo de la transparencia 15 de la clase anterior).
- Vamos a permitirnos escribir $=$ en lugar de $\equiv$. Por eso, siempre que veamos $a = b$ lo interpretaremos no como "a sintácticamente igual a b" sino como "a semánticamente equivalente a b".

(15) Elección de generadores y observadores

- Conviene que el conjunto de generadores y el de observadores sean *minimales*.
- Si los generadores no lo fuesen, se corre el riesgo de producir inconsistencias.
- Ídem con los observadores. Además, la redundancia atenta contra la claridad.
- Cómo elegirlos: ésta es la parte complicada. Veamos una herramienta que nos va servir para eso, llamada *igualdad observacional*.
- Antes, tratemos de entender de dónde proviene la dificultad. Repasemos el caso del restaurant.

(16) El pasaje de las expresiones humanas a las especificaciones

- Los lenguajes naturales son ambiguos, desde varios puntos de vista.
  - “Se calcula el índice de desempleo”. ¿Cómo se define un desempleado? (términos mal definidos)
  - “En el mismo texto pero más adelante”. *Más adelante*, ¿significa número de página menor o mayor? (polisemia).
  - “Para transportar (las cosas de [un lugar de África) a (otro de Rumania)] hace falta permiso”. ¿Cuándo hace falta permiso? (ambigüedad).
- Dicho de otra manera, tienen una *semántica difusa*.
- Los lenguajes (formales) de especificación son rigurosos, o dicho de otra manera, tienen una *semántica precisa*.

⚠ A esta diferencia se la conoce como *semantic gap* o *brecha semántica*.

(17) El pasaje de las expresiones humanas a las especificaciones (cont.)

- Es un paso importante porque una vez que tenemos una especificación formal existen herramientas capaces de automatizar todo el resto del proceso (al menos para casos pequeños).

⚠ El primer paso, es siempre "manual", trabajoso y subjetivo.

(18) La brecha semántica

- Este salto es un tema de complejidad no menor.
- La ingeniería del software dedica no pocos esfuerzos a él.
- En la materia lo encararemos de manera simplificada.
- En la práctica real del desarrollo de software, la etapa de análisis se trata de un proceso interactivo en el que el especificador interroga a los expertos del dominio y otros interesados para dilucidar las ambigüedades.
- En la materia, nos manejaremos con textos sencillos y los docentes tendremos ese rol durante la ejercitación.
- Como primer paso para vincular los mundos informales y formales, utilizaremos una herramienta llamada *igualdad observacional*.

(19) Igualdad observacional (como herramienta)

⚠ La igualdad observacional es un predicado entre instancias del tipo que nos dice cuándo son iguales (desde el punto de vista de su comportamiento).

- Notemos que $Ag(1, Ag(2, \emptyset))$ es sintacticamente distinto a $Ag(2, Ag(1, \emptyset))$, aunque desde el punto de vista de comportamiento nos gustaría que sean iguales.
- Para escribir la igualdad observacional utilizaremos ciertas funciones para ver detalles particulares de las instancias. Esas funciones son los observadores básicos.

⚠ Es decir, la igualdad observacional nos permite
  - deducir cuáles son los observadores necesarios,
  - explicitar cómo se combinan en un único predicado.

(20) Igualdad observacional (formalmente)

- La $=_{\text{obs}}$ es un predicado del metalenguaje, y como tal no lo podemos usar en los axiomas.
- ¿Y qué es un *metalenguaje*?
- Dentro del lenguaje podemos usar $\equiv$.
- Nosotros vamos a escribir la igualdad observacional como una forma de deducir los observadores y vamos a utilizar $\equiv$ (o $=$, como dijimos antes) en los axiomas.
- Y no le vamos a dar importancia a la diferencia teórica entre ambas.

(21) Congruencia

- Imaginemos el TAD NEGOCIO:

TAD NEGOCIO

generadores

inaugurar : $\longrightarrow$ negocio

vender : negocio $\times$ producto $\longrightarrow$ negocio

observadores básicos

total_vendido : negocio $\longrightarrow$ nat

otras operaciones

ventas : negocio $\longrightarrow$ secu(producto)

Fin TAD

- De acuerdo con esa especificación dos negocios van a ser iguales si su total vendido coincide.
- Pero podría suceder que si miramos el detalle de las ventas, difieran.
- ¿Qué está pasando?

(22) Congruencia (cont.)

- ⚠ Formalmente, la función ventas() rompe la congruencia. Ie, diferencia elementos que quedan en la misma clase de equivalencia de acuerdo con la igualdad observacional (o lo que es lo mismo, de acuerdo con los observadores básicos).
- ⚠ Recordemos que una función f es congruente con respecto a una relación de equivalencia $\sim$ ssi $(\forall x, y)\ (x \sim y \iff f(x) \sim f(y))$.
- Desde un punto de vista de modelado, o bien desaparece ventas() o bien se vuelve observador básico, y por ende miembro de la igualdad observacional.
- Si no hacemos eso, se genera una "inconsistencia": hay instancias del TAD NEGOCIO que son equivalentes según los observadores básicos, pero que son diferenciables de acuerdo a los axiomas efectivamente escritos.

(23) ¿Qué significa modelar?

- El de los TADs es un lenguaje lógico, y como tal, se busca que tenga un modelo: un conjunto para *interpretar* el género y funciones matemáticas para *interpretar* las operaciones.
- Sin embargo, cuando nos referimos a modelar, pensamos en la realidad como nuestro lenguaje y en encontrar un TAD que la capture.
- ¿Cómo aprendemos a pasar de la realidad al TAD? Con la práctica. Tenemos que resolver los ejercicios. Además, en las clases prácticas vamos a ir viendo algunas ideas.

(24) Repaso y perspectiva

- Vimos el TAD Conjunto (¡paramétrico!).
- Además de varios conceptos importantes.
  - Abstracción.
  - Modularidad.
  - Brecha semántica.
  - Igualdad observacional.
  - Congruencia.
- Seguiremos con:
  - Implementaciones de pilas y colas.
  - Memoria dinámica y punteros.

(25) Bibliografía

⚠ Ojo, bibliografía avanzada.

- Bernot, G., Bidoit, M., and Knapik, T. 1995. *Observational specifications and the indistinguishability assumption*. Theor. Comput. Sci. 139, 1-2 (Mar. 1995), 275-314. DOI= http://dx.doi.org/10.1016/0304-3975(94)00017-D
- Guttag, J. V. and Horning, J. J. 1993. *Larch: Languages and Tools for Formal Specification*. Springer-Verlag New York, Inc.
- *Abstraction and Specification in Software Development*, Barbara Liskov y John Guttag, MIT Press, 1986.
- *Fundamentals of Algebraic Specification 1. Equations and Initial Semantics.* H. Ehrig y B. Mahr Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, Germany, 1985.
- *Programación Metódica*, José Luis Balcázar, McGraw-Hill, 1993.