Rep y Abs: Soluciones

Algoritmos y Estructuras de Datos 2

1. Consejos para escribir el invariante

- Escribirlo primero en castellano y luego pasarlo a lógica (y relacionar ambas partes con números). De hecho, esto se pide explícitamente en algunos ejercicios.
- Tratar de resolver un predicados lógico de entrada con un $\Leftrightarrow$ ("si y sólo si") puede trabarnos y hacernos incurrir en errores. Usar el $\Leftrightarrow$ con cuidado y verificar que las dos implicaciones $\Rightarrow$ y $\Leftarrow$ sean equivalentes al $\Leftrightarrow$.
- Tener en cuenta (tipo checklist) que el invariante debe abarcar estos aspectos (notar que algunos de ellos se solapan):
  - Coherencia en la información redundante: Hay que chequear que distintos campos que proveen la misma información no se contradigan entre sí. Por ejemplo, en la estructura elegida para el TAD Conjunto en rango de la clase pasada, la longitud del rango del conjunto puede obtenerse tanto a partir del tamaño del arreglo e.elems como a partir del resultado de e.upper $-$ e.upper $+ 1$. Luego, en el invariante hay que pedir que estas dos formas de obtener la longitud no se contradigan. Es decir, que e.upper $-$ e.upper $+ 1$ $=$ tam(e.elems).
  - Restricciones del TAD: Hay que chequear que se vean reflejadas en la estructura de representación. Por ejemplo, en conjunto en rango el TAD indica que no se puede crear un conjunto con el límite inferior del rango mayor al límite superior. Entonces en el invariante hay que pedir que e.lower $\leq$ e.upper.
  - Decisiones de diseño: Hay que chequear las restricciones a la estructura de representación que no provengan de un chequeo de coherencia o de restricciones especificadas en el TAD. Por ejemplo, si decidimos implementar conjunto con una secuencia sin repetidos, entonces en el invariante debemos chequear que la secuencia, efectivamente, no tenga repetidos. La necesidad de hacer este chequeo no puede deducirse del TAD, y tampoco tiene que ver con un chequeo de coherencia de información redundante.

2. Palíndromos

Los palíndromos son aquellas palabras que pueden leerse al derecho o al revés. El siguiente TAD describe a los palíndromos:

TAD PALÍNDROMO(α)

observadores básicos

ver : palindromo(α) $\longrightarrow$ secu(α)

generadores

medio	: α	$\longrightarrow$	palindromo(α)
medioDoble	: α	$\longrightarrow$	palindromo(α)
agregar	: $\alpha \times$ palindromo(α)	$\longrightarrow$	palindromo(α)

axiomas

ver(medio(a))	$\equiv$	a $\bullet$ <>
ver(medioDoble(a))	$\equiv$	a $\bullet$ a $\bullet$ <>
ver(agregar(a,p))	$\equiv$	a $\bullet$ (ver(p) $\circ$ a)

Fin TAD

Se propone la siguiente estructura de representación:

`polindromo` **se representa con** `estr`

donde `estr` es `tupla`(*long*: `nat`, *palabra*: `secu(`α`)`)

dónde *palabra* representa el palíndromo completo.
Se pide:

a) Definir el invariante de representación y la función de abstracción.

b) Rehacer los ítems anteriores si el campo *palabra* en lugar de la palabra completa guardamos sólo la mitad inicial de la palabra (redondeando hacia arriba).

2.1. Invariante de representación en castellano

1. e.palabra tiene que ser un palíndromo. (decisión de diseño: el enunciado nos dice que (por alguna razón) se decide guardar en e.palabra a la secuencia asociada al palíndromo. Luego, dicha secuencia deberá ser un palíndromo)
2. La longitud de e.palabra tiene que coincidir con e.long. (redundancia: ¿Dónde esta la redundancia de información entre e.palabra y e.long? e.long nos dice la longitud del palíndromo, pero esa información también nos la provee e.palabra (simplemente tenemos que fijarnos su longitud). Es decir, que la parte de la información en la que ambos campos se "pisan" es la longitud del palíndromo. Sobre esa parte, entonces, tenemos que pedir que los dos campos mantengan la coherencia)
3. La longitud de e.long tiene que ser mayor a cero. (restricción del tad: un palíndromo no puede tener longitud cero.)

2.2. Invariante de representación

Rep : estr $\longrightarrow$ boolean
Rep(e) $\equiv$ true $\iff$ e.palabra = reverso(e.palabra) $\wedge$ long(e.palabra) = e.long $\wedge$ e.long > 0

2.3. Función de abstracción

Abs : estr e $\longrightarrow$ palíndromo $\{\text{Rep}(e)\}$
$(\forall e : \text{estr})$ Abs(e) $=_{\text{obs}}$ p: palíndromo | ver(p) = e.palabra

2.4. Cómo se modifican los ítems anteriores si sólo guardamos la mitad de la palabra?

1. e.palabra ya no tiene que ser (necesariamente) un palíndromo. (decisión de diseño: antes teníamos una decisión de diseño que decía que e.palabra guardaba la secuencia entera asociada al palíndromo. Por lo tanto, teníamos que asegurarnos de que e.palabra fuera, efectivamente, un palíndromo. Ahora guardamos sólo la mitad la secuencia entera y no hace falta que una secuencia sea un palíndromo para ser la mitad de otro palíndromo)

2. e.long tiene que ser el doble (o el doble menos uno) de la longitud de e.palabra. (redundancia: la redundancia sigue estando en la información acerca de la longitud del palíndromo (en una parte de ella). Si bien ahora la longitud del palíndromo no se puede deducir directamente de e.palabra, dicho campo todavía nos dice algo acerca de la longitud (justamente, que sera el doble o el doble menos uno de la longitud de e.palabra). Respecto a eso, tiene que seguir manteniendo la coherencia con la información que nos da e.longitud)

2.4.1. Invariante de representación

$\text{Rep} : \text{estr} \longrightarrow \text{boolean}$
$\text{Rep}(e) \equiv \text{true} \iff \neg\ \text{vacía?}(e.\text{palabra}) \wedge_{\text{L}} (2{\times}\text{long}(e.\text{palabra}) = e.\text{long} \vee 2{\times}\text{long}(e.\text{palabra}) - 1 = e.\text{long})$

2.4.2. Función de abstracción

$\text{Abs} : \text{estr}\ e \longrightarrow \text{palíndromo} \qquad \{\text{Rep}(e)\}$
$(\forall e : \text{estr})\ \text{Abs}(e) =_{\text{obs}} \text{p: palíndromo} \mid (2{\times}\text{long}(e.\text{palabra}) = e.\text{long} \wedge \text{ver}(p) = e.\text{palabra}\ \&\ \text{reverso}(e.\text{palabra})) \vee$
$(2{\times}\text{long}(e.\text{palabra}) - 1 = e.\text{long} \wedge \text{ver}(p) = e.\text{palabra}\ \&\ \text{reverso}(\text{com}(e.\text{palabra})))$

Comentarios:

- Notar que $2{\times}\text{long}(e.\text{palabra}) - 1$ no se indefine, pues pedimos que se cumpla $\text{Rep}(e)$, y en Rep habíamos pedido $\neg$ vacía?(e.palabra).

- Notar que ahora la secuencia asociada al palíndromo la construimos usando información de ambos campos de la estructura (antes sólo usábamos e.palabra, mientras que e.long era sólo información redundante)

3. Diseño - 1P 1C 2015

La siguiente especificación modela un castillo donde conviven piratas y ninjas. Con frecuencia arriban al castillo nuevos piratas y ninjas, que nunca mueren ni se van. Por supuesto, cada tanto surgen peleas, que por tradición ancestral son siempre entre un pirata y un ninja. Los piratas y los ninjas se identifican con naturales unívocos: no hay dos piratas, ni dos ninjas, ni un pirata y un ninja que se identifiquen con el mismo número.

TAD CASTILLO

observadores básicos

piratas	: castillo	$\longrightarrow$	conj(nat)	
ninjas	: castillo	$\longrightarrow$	conj(nat)	
cantPeleas	: castillo c $\times$ nat p $\times$ nat n	$\longrightarrow$	nat	$\{p \in \text{piratas}(c) \land n \in \text{ninjas}(c)\}$

generadores

crear	:	$\longrightarrow$	castillo	
llegaPirata	: castillo c $\times$ nat p	$\longrightarrow$	castillo	$\{p \notin (\text{piratas}(c) \cup \text{ninjas}(c))\}$
llegaNinja	: castillo c $\times$ nat n	$\longrightarrow$	castillo	$\{n \notin (\text{piratas}(c) \cup \text{ninjas}(c))\}$
pelean	: castillo c $\times$ nat p $\times$ nat n	$\longrightarrow$	castillo	$\{p \in \text{piratas}(c) \land n \in \text{ninjas}(c)\}$

axiomas

piratas(crear)	$\equiv \emptyset$	ninjas(crear)	$\equiv \emptyset$
piratas(llegaPirata(c, p))	$\equiv$ Ag(p, piratas(c))	ninjas(llegaPirata(c, p))	$\equiv$ ninjas(c)
piratas(llegaNinja(c, n))	$\equiv$ piratas(c)	ninjas(llegaNinja(c, n))	$\equiv$ Ag(n, ninjas(c))
piratas(pelean(c, p, n))	$\equiv$ piratas(c)	ninjas(pelean(c, p, n))	$\equiv$ ninjas(c)

cantPeleas(llegaPirata(c, p'), p, n) $\equiv$ **if** $p = p'$ **then** 0 **else** cantPeleas(c, p, n) **fi**
cantPeleas(llegaNinja(c, n'), p, n) $\equiv$ **if** $n = n'$ **then** 0 **else** cantPeleas(c, p, n) **fi**
cantPeleas(pelean(c, p', n'), p, n) $\equiv$ **if** $p = p' \land n = n'$ **then** 1 **else** 0 **fi** + cantPeleas(c, p, n)

Fin TAD

Para representar el TAD CASTILLO se decidió utilizar la siguiente estructura:

castillo **se representa con** estr, donde
estr es tupla ⟨ *piratas*: conj(nat),
ninjas: conj(nat),
rivalesQueTuvo: dicc(nat, conj(nat)),
historialPeleas: secu(tupla ⟨p : nat, n : nat⟩) ⟩

donde *piratas* y *ninjas* representan los conjuntos de identificadores de piratas y ninjas, respectivamente, *rivalesQueTuvo* asocia a cada peleador (tanto piratas como ninjas, ya que todos los identificadores son distintos) con el conjunto de todos los rivales contra los que peleó al menos una vez, e *historialPeleas* tiene la secuencia de parejas ⟨pirata, ninja⟩ que se entreveraron en una pelea, en el orden en que éstas sucedieron.

a) Escribir en castellano el invariante de representación.

b) Escribir formalmente el invariante de representación.

c) Escribir formalmente la función de abstracción.

3.1. Solución

Rep(e) $= 1 \wedge 2 \wedge_L 3 \wedge_L 4 \wedge 5 \wedge_L 6 \wedge 7$

1) No hay Piratas que sean Ninjas (y viceversa):
$e.piratas \cap e.ninjas = \emptyset$

2) Todas las claves de e.RivalesQueTuvo son piratas o ninjas (y viceversa):
$claves(e.RivalesQueTuvo) = e.piratas \cup e.ninjas$

3) Los rivales de un pirata son ninjas y viceversa:
$(\forall p : Nat)(p \in e.piratas \Rightarrow_L obtener(p, e.RivalesQueTuvo) \subseteq e.ninjas) \wedge$
$(\forall n : Nat)(n \in e.ninjas \Rightarrow_L obtener(n, e.RivalesQueTuvo) \subseteq e.piratas)$

Nota: Sé que n y p están definidos en $e.RivalesQueTuvo$ por cláusula 2).

4) Reciprocidad de rivales en e.RivalesQueTuvo:
$(\forall i : Nat)(def?(i, e.RivalesQueTuvo) \Rightarrow_L$
$(\forall j : Nat)(j \in obtener(i, e.RivalesQueTuvo) \Rightarrow_L i \in obtener(j, e.RivalesQueTuvo)))$

Nota: Sé que m está definido en $e.RivalesQueTuvo$ por cláusula 2) y 3).

5) Tuplas válidas en e.HistorialPeleas: un pirata y un ninja :
$(\forall t :< Nat, Nat >)(esta?(t, e.HistorialPeleas) \Rightarrow t.p \in e.piratas \wedge t.n \in e.ninjas)$

6) Las peleas de e.HistorialPeleas figuran correctamente en e.RivalesQueTuvo:
$(\forall t :< Nat, Nat >)(esta?(t, e.HistorialPeleas) \Rightarrow$
$t.p \in obtener(n, e.RivalesQueTuvo) \wedge t.n \in obtener(p, e.RivalesQueTuvo))$

Nota: Sé que $t.p$ y $t.n$ están definidos en $e.RivalesQueTuvo$ por 2) y 5).

7) Para cada luchador, los rivales que figuran en e.RivalesQueTuvo deben tener su pelea correspondiente en e.HistorialPeleas (básicamente la vuelta de 6):
$(\forall n : Nat)(n \in e.ninjas \Rightarrow_L$
$(\forall p : Nat)(p \in obtener(n, e.RivalesQueTuvo) \Rightarrow$
$(\exists t :< Nat, Nat >)(\pi_1(t) == p \wedge \pi_2(t) == n \wedge esta?(t, e.HistorialPeleas))))$

Nota: Sé que n está definido en $e.RivalesQueTuvo$ por cláusula 2).

4. Abs

Abs(e): estr e → Castillo c $\{Rep(e)\}$
Abs(e) $\equiv c : Castillo \mid$
$piratas(c) =_{obs} e.piratas \wedge$
$ninjas(c) =_{obs} e.ninjas \wedge_L$
$(\forall n, p : Nat)(n \in e.ninjas \wedge p \in e.piratas \Rightarrow_L$
$cantPeleas(c, p, n) = contarPeleas(e.HistorialPeleas, p, n))$

contarPeleas: secu(<Nat, Nat>) x Nat p x Nat p → nat
contarPeleas(s,p,n) ≡ if(vacia?(s)) then 0 else
(if($\pi_1(prim(s)) == p \wedge \pi_2(prim(s)) == n$) then 1 else 0) + contarPeleas(fin(s), p, n)

Errores Comunes:

- Comerse cláusula 7 y sólo hacer 6 (o vice versa)
- Poner $\wedge_L$ en lugar de $\Rightarrow_L$
- Comerse cláusula 4: la rivalidad debe ser simétrica