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

Rep : estr $\longrightarrow$ boolean
Rep(e) $\equiv$ true $\iff$ $\neg$ vacía?(e.palabra) $\wedge_{\text{L}}$ (2$\times$long(e.palabra) = e.long $\vee$ 2$\times$long(e.palabra) $-$ 1 = e.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?$(e.\text{palabra})$.
- Notar que ahora la secuencia asociada al palíndromo la construimos usando información de ambos campos de la estructura (antes sólo usábamos $e.\text{palabra}$, mientras que $e.\text{long}$ era sólo información redundante)

3. Sistema de seguimiento de maratones

Considerar la siguiente especificación de un sistema que realiza el seguimiento de varias maratones. Las maratones se enumeran por números naturales desde el 0 en adelante. De cada maratón solo se recuerdan los participantes y el orden en el que llegaron a la meta. Los participantes de las maratones se identifican por su número de DNI que es un NAT.

TAD MARATONES

observadores básicos

cantMaratones	: maratones	$\longrightarrow$	nat	
ranking	: maratonesm $\times$ nat i	$\longrightarrow$	secu(nat)	$\{i <$ cantMaratones$(m)\}$

generadores

iniciarTemporada	:	$\longrightarrow$	maratones	
nuevaMaraton	: maratones $\times$ secu(nat) s	$\longrightarrow$	maratones	$\{\neg$ vacía?$(s) \wedge$ sinRepetidos$(s)\}$

axiomas

cantMaratones(iniciarTemporada)	$\equiv$	0
cantMaratones(nuevaMaraton(m, s))	$\equiv$	1+cantMaratones(m)
ranking(nuevaMaraton(m, s),i)	$\equiv$	**if** $i + 1 =$ cantMaratones(m) **then** s **else** ranking(m, i) **fi**

Fin TAD

sinRepetidos es una función booleana que devuelve true si y sólo si la secuencia no contiene elementos repetidos

Se decidió utilizar la siguiente estructura para representar el TAD.

Maratones **se representa con** estr, donde

estr es tupla $\langle$*ordenPorIndice*: dicc(nat,secu(nat)),
personasEnPosicion: dicc(nat,conj(nat)),
participantes: conj(nat)$\rangle$

donde *ordenPorIndice* asocia cada índice válido de una maratón con la secuencia de personas en el orden en que llegaron a la meta. *personasEnPosicion* asocia a cada posición el conjunto de participantes que llegaron en al menos una maratón en esa posición. Solo se definen las posiciones en las que algún participante llegó alguna vez (i.e., no hay definiciones que sean el conjunto vacío). La posición del ganador de una maratón, que se obtiene haciendo prim de la secuencia correspondiente, es 0, y los siguientes se enumeran con naturales consecutivos. *participantes*, por último, tiene el conjunto de todas las personas que participaron alguna vez en alguna maratón.

a) Escribir en castellano el invariante de representación.

b) Escribir formalmente el invariante de representación.

c) Escribir formalmente la función de abstracción.

3.1. Invariante de representación en castellano

1. Si un participante aparece en una secuencia en ordenPorIndice, aparece también en un conjunto de personasEnPosicion, y dicho conjunto está definido para la misma posición en la que el participante aparecía en la secuencia. (redundancia: ¿Dónde esta la redundancia de información entre e.personasEnPosición y e.ordenPorIndice? En e.personasEnPosicion es posible ver, dada una posición, todas las personas que alguna vez llegaron en esa posición. Pero esa información también podemos obtenerla mirando las distintas secuencias de llegada en e.ordenPorIndice. Por ejemplo, para conocer todas las personas que alguna vez llegaron en la posición 0 (ganadores), basta con mirar la primera posición de cada una de las secuencias definidas en e.ordenPorIndice. Notar que e.ordenPorIndice nos da más información que e.personasEnPosición (por ejemplo, también nos dice qué personas corrieron en cada carrera). Sin embargo, la parte de la información en la ambos campos se "pisan" es la de que indica, para cada posición, las personas que alguna vez llegaron en esa posición. Luego, habrá que poner las restricciones necesarias para que ambos campos sean coherentes en cuanto a eso)

2. Si un participante aparece en el conjunto de una posición dada en personasEnPosicion, entonces aparece en alguna secuencia en ordenPorIndice en dicha posición. (redundancia: corresponde al mismo chequeo que el ítem anterior)

3. Si un participante aparece en una secuencia en ordenPorIndice, entonces está en participantes. (redundancia: en este caso, chequeamos la coherencia respecto a la parte de la información en la que se pisan e.ordenPorIndice y e.participantes: el conjunto de personas que alguna vez participaron en una maratón)

4. Si un participante está en participantes, entonces aparece en una secuencia en ordenPorIndice. (redundancia: corresponde al mismo chequeo que el caso anterior)

5. Toda secuencia definida en ordenPorIndice debe ser no vacía y no debe tener elementos repetidos. (restricciones del TAD: ver la operación nuevaMaraton en el TAD)

6. Las claves de ordenPorIndice (los índices de las maratones) no pueden saltearse números. (restricciones del TAD: asi lo indican los axiomas del observador ranking del TAD)

7. Los conjuntos definidos en personasEnPosicion no pueden ser vacíos. (decisión de diseño: especificado en el enunciado del problema)

Comentarios

- ¿Hace falta pedir que los participantes que aparecen en participantes sean los mismos que aparecen en personasEnPosición? Rta: No, ya está implícitamente pedido al pedir que los participantes de ordenPorIndice y personasEnPosicion coincidan y que los de ordenPorIndice y participantes coincidan.

3.2. Invariante de representación

Rep : estr $\longrightarrow$ boolean

Rep(e) $\equiv$ true $\Longleftrightarrow$

(1 y 2)

$(\forall d, p$: nat)

$((\exists m$:nat) (def?(m, e.ordenPorIndice) $\wedge_{\text{L}}$

está?(d, obtener(m, e.ordenPorIndice)) $\wedge_{\text{L}}$

posicion(d, obtener(m, e.ordenPorIndice)) $= p$)

$\Leftrightarrow$

def?(p, e.personasEnPosicion) $\wedge_{\text{L}}$ $d \in$ obtener(p, e.personasEnPosicion))

$\wedge$

(3 y 4)

$(\forall d$:nat) (($\exists m$: nat)(def?(m, e.ordenPorIndice) $\wedge_{\text{L}}$ está?(d, obtener(m, e.ordenPorIndice)))

$\Leftrightarrow$

$d \in e$.participantes)

$\wedge$

(5 y 6)

$(\forall m$:nat)(def?(m, e.ordenPorIndice) $\Rightarrow_{\text{L}}$ $\neg$ vacía?(obtener(m, e.ordenPorIndice)) $\wedge$

sinRepetidos(obtener(m, e.ordenPorIndice)) $\wedge$ ($m = 0 \vee_{\text{L}}$ def?($m - 1$, e.ordenPorIndice)))

$\wedge$

(7)

$(\forall p$:nat)(def?(p, e.personasEnPosicion) $\Rightarrow_{\text{L}}$ $\neg$ $\emptyset$?(obtener(p, e.personasEnPosicion)))

posicion : nat n $\times$ secu(nat) s $\longrightarrow$ nat {está?(n, s)}

posicion(n, s) $\equiv$ **if** prim(s) $= n$ **then** 0 **else** 1 + posicion(fin(s)) **fi**

Comentarios

- Al hacer auxiliares hay que tener cuidado de poner las restricciones que hagan falta para que la axiomatización sea correcta. Además, hay que revisar que no hagan faltas guardas con "luegos" para proteger dichas restricciones al usar las auxiliares. En el invariante anterior, por ejemplo, escribimos posicion(d, obtener(m, e.ordenPorIndice)) $= p$ después de un $\wedge_{\text{L}}$, y antes del $\wedge_{\text{L}}$ chequeamos que se cumpliera la restricción de la operación posición: está?(d, obtener(m, e.ordenPorIndice))

3.3. Función de abstracción

$$\begin{array}{l} \text{Abs} : \text{estr } e \longrightarrow \text{maratones} \qquad \{\text{Rep}(e)\} \\ (\forall e : \text{estr})\ \text{Abs}(e) =_{\text{obs}} \text{m: maratones} \mid \text{cantMaratones}(m) = \#(\text{claves}(e.\text{ordenPorIndice})) \wedge_{\text{L}} (\forall i\text{: nat})(i < \text{cantMaratones}(m) \\ \qquad \Rightarrow_{\text{L}} \text{ranking}(m,\ i) = \text{obtener}(i,\ e.\text{ordenPorIndice})) \end{array}$$

Comentarios

- ¿Por qué no hace falta preguntar si def?(i, e.ordenPorIndice) antes de hacer obtener(i, e.ordenPorIndice)? Rta: porque antes ya habíamos pedido algunas cosas:
  - en el antecedente del $\Rightarrow_{\text{L}}$ pedimos que $i <$ cantMaratones(m)
  - antes del $\wedge_{\text{L}}$ pedimos que cantMaratones(m) = #(claves(e.ordenPorIndice)
  - en el Rep pedimos que todas las claves de de e.ordenPorIndice fueran consecutivas, empezando en 0.

4. Torneo de Jiggly Ball

Considerar la siguiente especificación de un sistema que realiza el seguimiento de varios partidos de un torneo de Jiggly Ball. Cada partido es entre 2 equipos (identificados con strings) y el ganador se lleva un punto. Puede haber una cantidad arbitraria de partidos entre cada par de equipos.

TAD TORNEO

observadores básicos

equipos	: torneo	$\longrightarrow$	conj(string)	
puntos	: torneo t $\times$ string e	$\longrightarrow$	nat	$\{e \in \text{equipos}(t)\}$
historial	: torneo t $\times$ string e	$\longrightarrow$	secu(string)	$\{e \in \text{equipos}(t)\}$

generadores

nuevoTorneo	: conj(string) c	$\longrightarrow$	torneo	$\{\neg\emptyset?(c)\}$
regPartido	: torneo t $\times$ string g $\times$ string p	$\longrightarrow$	torneo	$\{g \in \text{equipos}(t) \wedge p \in \text{equipos}(t) \wedge g \neq p\}$

axiomas

equipos(nuevoTorneo(c))	$\equiv$	c
equipos(regPartido(t, g, p))	$\equiv$	equipos(t)
puntos(nuevoTorneo(c),e)	$\equiv$	0
puntos(regPartido(t, g, p),e)	$\equiv$	puntos(t, e) + **if** $e = g$ **then** 1 **else** 0 **fi**
historial(nuevoTorneo(c),e)	$\equiv$	<>
historial(regPartido(t, g, p),e)	$\equiv$	**if** $e = g$ **then** $p \bullet$ historial(t,e) **else** **if** $e = p$ **then** $g \bullet$ historial(t,e) **else** historial(t, e) **fi** **fi**

Fin TAD

Se decidió utilizar la siguiente estructura para representar el TAD.

Torneo **se representa con** estr, donde

estr es tupla $\langle$ *ranking*: secu(tupla $\langle$ *ptos*: nat, *eq*: string $\rangle$),
jugados: secu(tupla $\langle$ *eq1*: string, *eq2*: string $\rangle$)
derrotas: dicc(string, multiconj(string)) $\rangle$

donde *ranking* tiene una entrada por cada equipo que haya jugado al menos un partido, ordenados decrecientemente por cantidad de puntos (entre los que tengan la misma cantidad, es válido cualquier orden), *jugados* dice todos los partidos que se jugaron en orden cronológico, y *derrotas* dice para cada equipo el multiconjunto de los que lo vencieron (cada vencedor aparece repetido la cantidad de veces que le haya ganado al equipo en cuestión). Notar que en *jugados* *eq1* y *eq2* no se asocian de una manera determinada con ganador y perdedor del partido, es decir, la secuencia solo registra los partidos qué se jugaron, pero sin saber quien fue el ganador.

a) Escribir en castellano el invariante de representación.

b) Escribir formalmente el invariante de representación.

c) Escribir formalmente la función de abstracción.

4.1. Invariante de representación en castellano

1. e.ranking no puede tener equipos repetidos. (decisión de diseño (un poco obvia): el enunciado dice "...UNA entrada por cada equipo...")
2. e.ranking debe estar ordenado decrecientemente por puntos. (decisión de diseño: aclarada en el enunciado)
3. e.derrotas debe tener al menos una definición. (restricción del tad: el constructor nuevoTorneo requiere que haya al menos un equipo en el conjunto que se le pasa (y claves(e.derrotas) es el único lugar donde siempre van a estar todos los equipos del torneo (incluso los que todavía no jugaron)))
4. Una tupla de e.jugados no puede tener dos veces al mismo equipo. (restricción del tad: el constructor regPartido requiere dos equipos diferentes. Notar que esto, en combinación con 7 nos asegura que
($\forall$ r:string) #(r, obtener(r, e.derrotas)) = 0 (es decir, un equipo no puede perder contra si mismo))

5. e.ranking debe tener entradas para todos aquellos equipos que hayan jugado al menos un partido, y sólo para esos. (decisión de diseño: aclarado en el enunciado)

6. Todos los equipos que aparecen en e.jugados deben aparecer como clave en e.derrotas. (restricción del tad: el constructor regPartido dice que dos equipos que no están en el torneo no pueden jugar (y ya vimos que claves(e.derrotas) tendrá el conjunto de todos los equipos del torneo))

7. La cantidad de veces que un equipo fue derrotado por otro, sumado a la cantidad de veces que aquel fue derrotado por éste, tiene que ser igual a la cantidad de partidos que ambos jugaron. (redundancia: Dónde esta la redundancia de información entre e.jugados y e.derrotas? En e.derrotas es posible ver, dado un equipo r_0, cuantas veces perdió con otro equipo r_1 y viceversa. Pero al darnos esta información, también nos está dando implícitamente la información de la cantidad de veces que ambos equipos jugaron, información que ya estaba contenida en e.jugados. De ahí viene la redundancia. Dicho de otra forma, e.jugados nos dice cuantas veces dos equipos r_0 y r_1 jugaron (también nos dice el orden en el que se dieron los partidos en el tiempo, pero eso no viene al caso), y e.derrotas también nos dice eso, sólo que agregando, además, la información de quién perdió con quién. La parte de la información en la ambos campos se "pisan" es la cantidad de partidos que jugaron dos equipos cualquiera. Luego, habrá que poner las restricciones necesarias para que ambos campos sean coherentes en cuanto a eso)

8. El puntaje de un equipo tiene que ser igual a la cantidad de partidos jugados menos la cantidad de derrotas que sufrió. (redundancia: Para cada equipo (que jugó al menos un partido), e.ranking nos dice la cantidad de puntos que tiene (es decir, la cantidad de partidos que ganó). Ahora bien, esa información ya nos la proveían (en conjunto) los campos e.jugados y e.derrotas. ¿Cómo se "distribuía" dicha información entre ambos campos? Dado un equipo, puedo saber cuantos partidos jugó mirando e.jugados. Además, puedo saber cuántos de esos partidos perdió, mirando e.derrotas. De la resta de ambos valores sale la cantidad de partido ganados. La parte de la información en la que e.ranking se "pisa" con los otros dos campos, es la que obtengo de dicha resta. Luego, habrá que poner las restricciones necesarias para que los tres campos sean coherentes respecto a esa información)

4.2. Invariante de representación

Rep : estr $\longrightarrow$ boolean
Rep(e) $\equiv$ true $\Longleftrightarrow$

(1 y 2)
sinEquiposRepetidos(e.ranking) $\wedge$ decrecientePorPuntos(e.ranking)
$\wedge$
(3)
$\neg\emptyset$(claves(e.derrotas))
$\wedge$
(4)
($\forall q$: string)
(cantidadEnfrentamientos(q, q, e.jugados) = 0)
$\wedge$
$\Big($(
(5 y 6)
($\forall q$: string)
((está?(q, segundaComponente(e.ranking)) $\Leftrightarrow$ cantidadJugados(q, e.jugados) $\geq$ 1)
$\wedge$
(cantidadDeJugados(q, e.jugados) $\geq$ 1 $\Rightarrow$ def?(q, e.derrotas)))
$\wedge$
(7)
($\forall q_0, q_1$: string)
(def?(q_0, e.derrotas) $\wedge$ def?(q_1, e.derrotas) $\Rightarrow_{\text{L}}$
#(q_0, obtener(q_1, e.derrotas)) + #(q_1, obtener(q_0, e.derrotas)) =
cantidadEnfrentamientos(q_0, q_1, e.jugados))
$\Big)$
$\wedge_{\text{L}}$
(8)
($\forall t$: tupla $\langle ptos$: nat, eq: string$\rangle$) (está?(t, e.ranking) $\Rightarrow_{\text{L}}$
t.ptos = cantidadJugados(t.eq, e.jugados) - #(obtener(t.eq, e.derrotas)))$\Big)$

sinEquiposRepetidos : secu(tupla(nat $\times$ string)) $\longrightarrow$ bool
sinEquiposRepetidos(s) $\equiv$ vacía?(s) $\vee_{\text{L}}$ ($\neg$ está?(prim(s).eq, segundaComponente(fin(s))) $\wedge$ sinEquiposRepetidos(fin(s)))

segundaComponente : secu(tupla($\beta \times \alpha$)) $\longrightarrow$ secu(α)
segundaComponente(s) $\equiv$ **if** vacía?(s) **then** <> **else** prim(s)$_2$ $\bullet$ segundaComponente(fin(s)) **fi**

decrecientePorPuntos : secu(tupla(nat $\times$ string)) $\longrightarrow$ bool
decrecientePorPuntos(s) $\equiv$ long(s) $\leq$ 1 $\vee_{\text{L}}$ (prim(s).ptos $\geq$ prim(fin(s)).ptos $\wedge$ decrecientePorPuntos(fin(s)))

cantidadEnfrentamientos : string $\times$ string $\times$ secu(tupla(string $\times$ string)) $\longrightarrow$ nat
cantidadEnfrentamientos(q_1, q_2, s) $\equiv$ **if** vacía?(s) **then**
0
else
cantidadEnfrentamientos(q_1, q_2, fin(s)) +
if (prim(s).eq1 = q_1 $\wedge$ prim(s).eq2 = q_2) $\vee$ (prim(s).eq1 = q_2 $\wedge$ prim(s).eq2 = q_1) **then** 1 **else** 0 **fi**
fi

cantidadJugados : string $\times$ secu(tupla(string $\times$ string)) $\longrightarrow$ nat

$$\begin{array}{l}\text{cantidadJugados}(q, s) \equiv \textbf{if } \text{vacía?}(s) \textbf{ then} \\ \quad 0 \\ \textbf{else} \\ \quad \text{cantidadJugados}(q, \text{fin}(s)) + \\ \quad \textbf{if } \text{prim}(s).\text{eq1} = q \vee \text{prim}(s).\text{eq2} = q \textbf{ then } 1 \textbf{ else } 0 \textbf{ fi} \\ \textbf{fi}\end{array}$$

Comentarios

- Notar que hace falta pedir que $((5 \wedge 6 \wedge 7) \wedge_{\text{L}} 8)$. El yluego es importante. Si no hacemos esto, el pedir que la tupla este en e.ranking no nos asegura que el equipo esté definido en e.derrotas (esa parte la asegura pedir $5 \wedge 6$) y, por otro lado, no podríamos asegurar que la resta que hacemos en 2 no se va a indefinir (esa parte la asegura pedir 7).

4.3. Función de abstracción

$$\begin{array}{l}\text{Abs} : \text{estr } e \longrightarrow \text{torneo} \qquad \{\text{Rep}(e)\} \\ (\forall e : \text{estr})\ \text{Abs}(e) =_{\text{obs}} \text{t: torneo} \mid \text{equipos}(t) = \text{claves}(e.\text{derrotas}) \wedge \\ \qquad (\forall q\text{: string})(q \in \text{equipos}(t) \Rightarrow_{\text{L}} \\ \qquad \text{puntos}(t, q) = \text{puntos}(e.\text{ranking}, q) \wedge \text{historial}(t, q) = \text{historial}(e.\text{jugados}, q))\end{array}$$

$$\begin{array}{l}\text{puntos} : \text{secu}(\text{tupla}(\text{nat} \times \text{string})) \times \text{string} \longrightarrow \text{nat} \\ \text{puntos}(s, q) \equiv \textbf{if } \text{vacía?}(s) \textbf{ then } 0 \textbf{ else if } \text{prim}(s).\text{eq} = q \textbf{ then } \text{prim}(s).\text{ptos} \textbf{ else } \text{puntos}(\text{fin}(s), q) \textbf{ fi fi}\end{array}$$

$$\begin{array}{l}\text{historial} : \text{secu}(\text{tupla}(\text{string} \times \text{string})) \times \text{string} \longrightarrow \text{secu}(\text{string}) \\ \text{historial}(<>, q) \equiv <> \\ \text{historial}(p \bullet s, q) \equiv \textbf{if } p.\text{eq1} = q \textbf{ then} \\ \qquad p.\text{eq2} \bullet \text{historial}(s, q) \\ \quad \textbf{else} \\ \qquad \textbf{if } p.\text{eq2} = q \textbf{ then } p.\text{eq1} \bullet \text{historial}(s, q) \textbf{ else } \text{historial}(s, q) \textbf{ fi} \\ \quad \textbf{fi}\end{array}$$