Guia 4 - TAD

April 2024

1. Ejercicio 1

TAD Racional {
obs Num:$\mathbb{Z}$
obs Den:$\mathbb{Z}$

`proc nuevoNumero` (in x: $\mathbb{Z}$, in y:$\mathbb{Z}$) : Racional
`requiere` $\{y \neq 0\}$
`asegura` $\{res.Num = x \ \wedge \ res.Den = y\}$

`proc suma` (inout x:Racional,in y:Racional)
`requiere` $\{x = x0\}$
`asegura` $\{x.Num/x.Den = x0.num/x0.Den + y.Num/y.Den\}$

`proc resta` (inout x:Racional,in y:Racional)
`requiere` $\{x = x0\}$
`asegura` $\{x.Num/x.Den = x0.num/x0.Den - y.Num/y.Den\}$

`proc multiplicacion` (inout x:Racional,in y:Racional)
`requiere` $\{x = x0\}$
`asegura` $\{x.Num/x.Den = x0.Num * y.Num/x0.Den * y.Den\}$

`proc division` (inout x:Racional,in y:Racional)
`requiere` $\{x = x0\}$
`asegura` $\{x.Num/x.Den = x0.Num * y.Den/x0.Den * y.Num\}$

}

2. Ejercicio 2

2.1. A)

TAD Rectangulo {
obs base:$\mathbb{R}$
obs altura:$seq\langle\mathbb{R}\rangle$
obs angulo:$\mathbb{R}$
obs centro:$seq\langle\mathbb{R}\rangle$

`proc nuevoRectangulo` (in b:$\mathbb{R}$, in h:$\mathbb{R}$, in angulo:$\mathbb{R}$, in centro:$seq\langle\mathbb{R}\rangle$) : Rectangulo
`requiere` $\{|centro| = 2\}$
`asegura` $\{res.base = b \ \wedge \ res.altura = h \ \wedge res.angulo = angulo \ mod \ 2\pi \ \wedge \ res.centro = centro\}$

`proc rotar` (inout r:Rectangulo , in angulo:$\mathbb{R}$)
`requiere` $\{r = r0\}$

asegura $\{r.base = r0.base \ \wedge \ r.altura = r0.altura \ \wedge \ r.centro = r0.centro \ \wedge$
$r.angulo = (r0.angulo + angulo) \ mod \ 2\pi\}$

proc trasladar (inout r:Rectangulo , in coord:$seq\langle\mathbb{R}\rangle$)
requiere $\{r = r0 \wedge |coord| = 2\}$
asegura $\{r.base = r0.base \ \wedge \ r.altura = r0.altura \ \wedge \ r.angulo = r0.angulo \ \wedge$
$r.centro = [r0.centro[0] + coord[0], r0.centro[1] + coord[1]]\}$

proc escalar (inout r:Rectangulo, in b:$\mathbb{R}$, in h:$\mathbb{R}$)
requiere $\{r = r0 \ \wedge \ b > 0 \ \wedge \ h > 0\}$
asegura $\{r.base = r0.base * b \ \wedge \ r.altura = r0.altura * h \ \wedge \ r.centro = r0.centro \ \wedge$
$r.angulo = r0.angulo\}$

proc area (in r:Rectangulo) : $\mathbb{R}$
requiere $\{True\}$
asegura $\{res = r.base * r.altura\}$

}

2.2. B)

TAD Esfera {
obs obs radio:$\mathbb{R}$
obs obs centro:$seq\langle\mathbb{R}\rangle$
obs rotacion:$seq\langle\mathbb{R}\rangle$

proc nuevaEsfera (in r:$\mathbb{R}$, c:$seq\langle\mathbb{R}\rangle$, in angulos:$seq\langle\mathbb{R}\rangle$) : Esfera
requiere $\{r > 0 \wedge |angulos| = 3\}$
asegura $\{res.radio = r \ \wedge \ res.centro = c \ \wedge$
$res.rotacion = [angulos[0] \ mod \ 2\pi, angulos[1] \ mod \ 2\pi, angulos[2] \ mod \ 2\pi]\}$

proc rotar (inout e:Esfera, in x:$\mathbb{R}$, in y:$\mathbb{R}$, in z:$\mathbb{R}$)
requiere $\{e = e0 \ \wedge \ |rotacion| = 3\}$
asegura $\{e.radio = e0.radio \ \wedge \ e.centro = e0.centro \ \wedge$
$e.rotacion = [e0.rotacion[0] + x \ mod \ 2\pi, \ e0.rotacion[1] + y \ mod \ 2\pi, \ e0.rotacion[1] + z \ mod \ 2\pi]\}$

proc trasladar (inout e:Esfera, in coord:$seq\langle\mathbb{R}\rangle$)
requiere $\{e = e0 \ \wedge \ |coord| = 3\}$
asegura $\{e.radio = e0.radio \ \wedge \ e.rotacion = e0.rotacion \ \wedge$
$e.centro = [e0.centro[0] + coord[0], e0.centro[1] + coord[1], e0.centro[2] + coord[2]]\}$

proc escalar (inout e:Esfera, in r:$\mathbb{R}$)
requiere $\{e = e0 \ \wedge \ r > 0\}$
asegura $\{e.rotacion = e0.rotacion \ \wedge \ e.centro = e0.centro \ \wedge \ e.radio = e0.radio * r\}$

}

3. Ejercicio 3

TAD DobleCola$\langle T\rangle${
obs dc:$seq\langle\mathbb{R}\rangle$

proc nuevaDobleCola () : DobleCola $\langle T\rangle$
asegura $\{res.dc = \langle\rangle\}$

proc encolarAdelante (inout s:DobleCola$\langle T \rangle$, $e : T$)
requiere $\{s = s0\}$
asegura $\{s.dc = concat(\langle e \rangle, s0.dc)\}$

proc encolarAtras (inout s:DobleCola$\langle T \rangle$, $e : T$)
requiere $\{s = s0\}$
asegura $\{s.dc = concat(s0.dc, \langle e \rangle)\}$

proc desencolar (inout s:DobleCola$\langle T \rangle$) : T
requiere $\{s = s0 \ \wedge \ s0.dc \neq \langle\rangle\}$
asegura $\{(|s0.dc| = 1 \ \wedge \ s.dc = \langle\rangle \ \wedge \ res = s0.dc[0]) \vee_L$
$(s.dc = concat(subseq(s0.dc, 0, medio(s0.dc)), subseq(s0.dc, (medio(s0.dc) + 1), |s0.dc|)) \wedge$
$res = s0.dc[medio(s0.dc)])\}$

aux medio (s:$seq\langle \mathbb{R} \rangle$) : $\mathbb{R}$ = if $|s|$ mod $2 \neq 0$ then $(|s|/2) - 0{,}5$ else $(|s|/2) - 1$ fi ;
}

4. Ejercicio 4

TAD Diccionario[K,V] {
obs data:dict[K,V]

proc diccionarioVacio () : Diccionario[K,V]
requiere $\{True\}$
asegura $\{res.data = \{\}\}$

proc definirHistorial (inout dick:Diccionario[K,V], in k:K , in v:V)
requiere $\{dick = d0\}$
asegura $\{(k \in d0.data \ \wedge \ dick.data = setKey(d0.data, k, concat(d0, \langle v \rangle)) \ \vee$
$(k \notin d0.data \ \wedge \ dick.data = setKey(d0.data, k, \langle v \rangle))\}$

}

5. Ejercicio 5

TAD ColaDePrioridad$\langle T \rangle${
obs cola:dict$\langle T, \mathbb{R} \rangle$

proc desapilarMax (inout c:ColaDePrioridad$\langle T \rangle$) : $\langle T \rangle$
requiere $\{c = c0\}$
asegura $\{(\forall i : \mathbb{Z})\ (0 \leq i < |res| \longrightarrow_L esPriMax(c0, res[i]) \ \wedge \ res[i] \notin c.cola \ \wedge$
$((\forall k : T)\ (k \in c.cola \longrightarrow_L k \in c0.cola)))\}$

pred esPriMax (c:ColaDePrioridad$\langle T \rangle, k : T$){
$k \in c \ \wedge \ (\forall k2 : T)\ (k2 \in c \longrightarrow_L c[k] \geq c[k2])$
}
}

6. Ejercicio 6

6.1. A)

TAD Dicksionario {

obs data:conj$\langle(K, V)\rangle$

proc `dicksionarioVacio` () : Dicksionario

`requiere` $\{True\}$

`asegura` $\{res.data = \{\}\}$

pred `esta` (dc:Dicksionario, k:K) {

$(\exists v : V)\ ((k, v) \in dc.data)$

}

proc `definir` (inout dc:Dicksionario, in k:K, in v:V)

`requiere` $\{dc = dc0\}$

`asegura` $\{(\neg esta(dc0.data, k)\ \wedge\ dc.data = dc0.data \cup (k, v))\ \vee\ (esta(dc0.data, k)\ \wedge$
$(\exists u : V)\ ((k, u) \in dc0.data\ \wedge\ dc.data = (dc0.data - (k, u)) \cup (k, v))\}$

proc `obtener` (in dc:Dicksionario,in k:K) : V

`requiere` $\{true\}$

`asegura` $\{(\exists i : \mathbb{Z})\ (0 \leq i < |dc.data|\ \wedge_L\ (dc.data[i][0] = k\ \wedge\ res = dc.data[i][1]))\}$

proc `borrar` (inout dc:Dicksionario, in k:K)

`requiere` $\{dc = dc0\ \wedge\ esta(dc0.data, k)\}$

`asegura` $\{(\exists v : V)\ ((k, v) \in dc0.data\ \wedge_L\ dc.data = dc0.data - (k, v))\}$

proc `length` (in dc:Dicksionario) : $\mathbb{Z}$

`requiere` $\{true\}$

`asegura` $\{res = |dc.data|\}$

6.2. B)

Esta resuelto en la teorica.

6.3. C)

TAD Pila {

obs pila:dict$\langle\mathbb{R}, T\rangle$

proc `nuevaPila` () : Pila

`requiere` $\{true\}$

`asegura` $\{res.pila = \{\}\}$

proc `apilar` (inout p:Pila, in k:T)

`requiere` $\{p = p0\}$

`asegura` $\{p.pila = setKey(p0.pila, |p0.pila|, k)\}$

proc `desapilar` (p:Pila) : T

`requiere` $\{p = p0\ \wedge\ p0.pila \neq \{\}\}$

asegura $\{res = p0.pila[|p.pila| - 1] \ \wedge \ p.pila = delKey(p0.pila, |p0.pila| - 1\}$

proc tope (in p:Pila) : T

requiere $\{p.pila \neq \{\}\}$

asegura $\{res = p.pila[|p.pila] - 1]\}$

}

6.4. D)

Esta resuelto en la teorica.

7. Ejercicio 7

7.1. A)

TAD Multiconjunto {

obs apariciones(e: T):$\mathbb{Z}$

proc nuevoConjunto () : Multiconjunto[T]

requiere $\{true\}$

asegura $\{(\forall e : T) \ (res.apariciones(e) = 0)\}$

proc agregar (inout m:Multiconjunto, in e:T)

requiere $\{m = m0\}$

asegura $\{(\forall t : T) \ (t \neq e \longrightarrow_L m.apariciones(t) = m0.apariciones(t) \ \wedge$
$m.apariciones(e) = m.apariciones(e) + 1)\}$

pred pertenece (m:Multiconjunto, e:T) {

$m.cantApariciones(e) > 0$

}

proc sacar (inout m:Multiconjunto,in e:T)

requiere $\{m = m0 \ \wedge \ e \in m.elems\}$

asegura $\{(\forall t : T) \ (t \neq e \longrightarrow_L m.apariciones(t) = m0.apariciones(t) \ \wedge$
$m.apariciones(e) = m.apariciones(e) - 1)\}$

proc multiplicidad (in m:Multiconjunto, in e:T) : $\mathbb{Z}$

requiere $\{e \in m\}$

asegura $\{res = m.apariciones(e)\}$

7.2. B)

TAD Multidick {

obs data:dict[K,$seq\langle T\rangle$]

proc nuevoDick () : Multidick[K,$seq\langle T\rangle$]

`requiere` $\{true\}$

`asegura` $\{res.data = \{\}\}$

`proc definir` (inout m:Multidick, k:K, v:T)

`requiere` $\{m = m0\}$

`asegura` $\{k \in m0.data \ \wedge \ m.data = setKey(m0.data, k, concat(m0.data[k], [v])) \ \vee k \notin m0.data \ \wedge \ m.data = setKey(m0.data, k, [v])\}$

`proc obtener` (in m:Multidick, in k:K) : $seq\langle T\rangle$

`requiere` $\{k \in m\}$

`asegura` $\{res = m.data[k]\}$

`proc borrar` (inout m:Multidick, in k:K)

`requiere` $\{m = m0 \ \wedge \ k \in m.data\}$

`asegura` $\{m.data[k] = subseq(m0, 0, |m0.data| - 1\}$

`proc borrarTodo` (inout m:Multidick, in k:K)

`requiere` $\{m = m0 \ \wedge \ k \in m.data\}$

`asegura` $\{m.data = delKey(m0.data, k)\}$

8. Ejercicio 8

TAD Contador {

obs datos: dict[T,$\mathbb{Z}$]

`proc contadorVacio` () : Contador

`requiere` $\{true\}$

`asegura` $\{res.datos = \{\}\}$

`proc incrementar` (inout c:Contador, in e:T)

`requiere` $\{c = c0 \ \wedge \ e \in c.datos\}$

`asegura` $\{c.datos[e] = setKey(c0.datos, e, c0.datos[e] + 1)\}$

`proc cantidad` (in c:Contador, in e:T) : $\mathbb{Z}$

`requiere` $\{e \in c.datos\}$

`asegura` $\{res = c.datos[e]\}$

9. Ejercicio 9

obs orden:$seq\langle T\rangle$

`proc incrementarSoloAlgunos` (inout c:Contador, in porcentaje:$\mathbb{R}$)

`requiere` $\{c = c0\}$

asegura $\{(\forall i : \mathbb{Z})\ (0 \leq i < cuantos(c0.datos, porcentaje) \longrightarrow_L$
$c.datos[c0.orden[i]] = c0.datos[orden[i]] + 1)\}$

asegura $\{((\forall j : \mathbb{Z})\ (cuantos(c0.datos, porcentaje) < j < |c0.orden| \longrightarrow_L$
$c.datos[c0.orden[j]] = c0.datos[c0.orden[j]])\}$

10. Ejercicio 10

10.1. A)

TAD CacheFIFO[K,V] {

obs cap:$\mathbb{Z}$

obs claves:$seq\langle K\rangle$

obs data:dict[K,V]

proc nuevoCache (cap:$\mathbb{Z}$) : CacheFIFO[K,V]

asegura $\{res.cap = cap\ \wedge\ res.claves = \{\}\ \wedge\ res.data = \{\}\}$

proc esta (in c:CacheFIFO, in k:K) : bool

requiere $\{true\}$

asegura $\{res = true \iff k \in c.data\}$

proc obtener (in c:CacheFIFO, in k:K) : V

requiere $\{k \in c.data\}$

asegura $\{res = c.data[k]\}$

proc definir (inout c:CacheFIFO,in k:K, in v:V)

requiere $\{c = c0\}$

asegura $\{k \in c0.claves \longrightarrow c.claves = c0.claves\ \wedge\ c.data = setKey(c0.data, k, v)\}$

asegura $\{k \notin c0.claves \wedge |c0.claves| < cap \longrightarrow c.claves = concat(c0.claves, k)\ \wedge$
$c.data = setKey(c0.data, k, v)\}$

asegura $\{k \notin c0.claves \wedge |c0.claves| = cap \longrightarrow c.claves = concat(subseq(c0.claves, 1, |c0.claves|, [k]) \wedge$
$c.data = setKey(delKey(c0.data, c0.data[c.claves[0]]), k, v)\}$

asegura $\{c.cap = c0.cap\}$

10.2. B)

TAD CacheLRU[K,(V,$\mathbb{R}$)] {

obs claves:$seq\langle K\rangle$

obs data:dict[K,(V,$\mathbb{R}$)]

obs cap :$\mathbb{N}$

proc nuevoCache (in cap:$\mathbb{N}$) : CacheLRU[K,(V,$\mathbb{R}$)]

requiere $\{true\}$

asegura $\{res.data = \{\}\ \wedge\ res.cap = cap\ \wedge\ |res.claves| = 0\}$

proc esta (in c:CacheLRU, in k:K) : bool

requiere $\{true\}$

asegura $\{res = true \iff k \in c.data\}$

proc obtener (in c:CacheLRU, in k:K) : V

requiere $\{k \in c.data\}$

asegura $\{res = c.data[k]\}$

proc definir (c:CacheLRU, in k:K,in v:V)

requiere $\{c = c0\}$

asegura $\{k \in c0.claves \longrightarrow (c.data = setKey(c0.data, k, (v, horaActual()) \ \wedge \ c.claves = c0.claves)\}$

si k esta en el diccionario , dejo todo como esta y actualizo su valor con el tiempo en que se agrego.

asegura $\{k \notin c0.claves \ \wedge \ |c0.claves| < c0.cap \longrightarrow c.data = setKey(c0.data, k, (v, horaActual()) \ \wedge$
$c.claves = concat(c0.claves, k))\}$

si k no esta en el dic pero todavia tiene capacidad , agrego k a la lista de claves y actualizo su valor con el tiempo que entro.

asegura $\{k \notin c0.claves \ \wedge \ |c0.claves| = cap \longrightarrow ((\exists i : \mathbb{Z}) \ (0 \leq i < |c0.claves| \ \wedge$
$antiguo(c0.data, c0.claves[i]) \wedge claves(c.claves, c0.claves, i, k) \wedge actualizo(c.data, c0.data, c0.claves[i], k, v))\}$

si k no esta en el dic pero no tengo mas capacidad , saco la clave mas antigua del dic y la lista de claves , luego actualizo con la nueva clave k.

pred antiguo (c0:dict[K,(V,$\mathbb{R}$)], k:K) {

$(\forall k' : T) \ (k' \in c0 \ \wedge \ k' \neq k \longrightarrow_L c0[k][1] < c0[k'][1])$

}

pred claves (c:$seq\langle T\rangle$, c0:$seq\langle T\rangle$, i:$\mathbb{Z}$, k:T) {

$i = |c0| - 1 \ \wedge \ c = concat(subseq(c0, 0, i), [k]) \ \vee_L$
$c = concat(concat(subseq(c0, 0, i), subseq(c0, i + 1, |c0|), [k]))$

}

pred actualizo (c:dict[K,(V,$\mathbb{R}$)],c0:dict[K,(V,$\mathbb{R}$)],ant:K,k:K,v:V) {

$c = setKey(delKey(c0, ant), k, (v, horaActual()))$

}

10.3. C)

TAD CacheTTL[K,(V,$\mathbb{R}$)]}

obs data:dict[K,(V,$\mathbb{R}$)]

proc nuevoCache () : CacheTTL[K,(V,$\mathbb{R}$)]

asegura $\{res.data = \{\}\}$

proc esta (in c:CacheTTL, in k:K) : bool

requiere $\{true\}$

asegura $\{res = true \iff k \in c.data\}$

proc obtener (in c:CacheTTL, in k:K) : (V,$\mathbb{R}$)

requiere $\{k \in c.data\}$

asegura $\{res = c.data[k]\}$

proc definir (inout c:CacheTTL, in k:K , in v:V,in ttl:$\mathbb{R}$)

requiere $\{c = c0 \ \wedge \ ttl > 0\}$

asegura $\{c.data = setKey(c0.data, k, (v, ttl))\}$

proc actualizar (inout c:CacheTTL)

requiere $\{c = c0\}$

asegura $\{(\forall k : T)\ (k \in c0.data \longrightarrow_L \text{if } c0.data[k][1] \leq horaActual() \text{ then } k \notin c.data \text{ else } k \in c.data \text{ fi}) \ \wedge \ (\forall k'' : T)\ (k'' \notin c0.data \longrightarrow_L k'' \notin c.data)\}$

11. Ejercicio 11

Coord es struct $\langle x : \mathbb{Z}, y : \mathbb{Z} \rangle$

TAD Robot {

obs pos:Coord

obs pasajes:$seq\langle Coord \rangle$

proc inicia (in posx:$\mathbb{Z}$,posy:$\mathbb{Z}$) : Robot

requiere $\{true\}$

asegura $\{res.pos =< x : posx,\ y : posy > \ \wedge res.pasajes = [< x : posx,\ y : posy >]\}$

proc arriba (inout r:Robot)

requiere $\{r = r0\}$

asegura $\{r.pos.x = r0.pos.x \ \wedge \ r.pos.y = r0.pos.y + 1 \ \wedge$
$r.pasajes = concat(r0.pasajes, [< x : r0.pos.x,\ y : r0.pos.y + 1 >])\}$

proc abajo (inout r:Robot)

requiere $\{r = r0\}$

asegura $\{r.pos.x = r0.pos.x \ \wedge \ r.pos.y = r0.pos.y - 1 \ \wedge$
$r.pasajes = concat(r0.pasajes, [< x : r0.pos.x,\ y : r0.pos.y - 1 >])\}$

proc izquierda (inout r:Robot)

requiere $\{r = r0\}$

asegura $\{r.pos.x = r0.pos.x - 1 \ \wedge \ r.pos.y = r0.pos.y \ \wedge$
$r.pasajes = concat(r0.pasajes, [< x : r0.pos.x - 1,\ y : r0.pos.y >])\}$

proc derecha (inout r:Robot)

requiere $\{r = r0\}$

asegura $\{r.pos.x = r0.pos.x + 1 \ \wedge \ r.pos.y = r0.pos.y \ \wedge$
$r.pasajes = concat(r0.pasajes, [< x : r0.pos.x + 1,\ y : r0.pos.y >])\}$

proc masDerecha (in r:Robot) : $\mathbb{Z}$

`requiere` $\{true\}$

`asegura` $\{(\exists j : \mathbb{Z})\ (0 \leq j < |r.pasajes|\ \wedge\ r.pasajes[j].x = res\ \wedge\ (\forall i : \mathbb{Z})\ (0 \leq i < |r.pasajes|\ \wedge\ i \neq j \longrightarrow_L r.pasajes[i].x \leq res))\}$

`proc cuantasVecesPaso` (in r:Robot, in c:Coord) : $\mathbb{Z}$

`requiere` $\{true\}$

`asegura` $\{res = \sum_{i=0}^{|r.pasajes|-1} \text{if } r.pasajes[i] = c \text{ then } 1 \text{ else } 0 \text{ fi}\}$

12. Ejercicio 12

TAD Vivero {

obs stock:dict[K,$\mathbb{R}$]

obs caja:$\mathbb{R}$

obs venta:dict[K,$\mathbb{R}$]

`proc inicio` (recursos:$\mathbb{R}$) : Vivero

`requiere` $\{recursos > 0\}$

`asegura` $\{res.caja = recursos\ \wedge\ res.stock = \{\}\ \wedge\ res.venta = \{\}\}$

`proc stockear` (inout viv:Vivero, in k:K,in cant:$\mathbb{N}$, in precio:$\mathbb{R}$) : $\mathbb{R}$

`requiere` $\{viv = viv0\ \wedge\ viv.caja > 0\}$

`asegura` $\{viv.caja > 0\ \wedge\ viv.caja = viv0.caja - cant * precio\}$

`asegura` $\{(k \in viv.stock\ \wedge viv.stock = setKey(viv0.stock, k, (viv0.stock[k] + cant))) \vee (k \notin viv.stock\ \wedge\ viv.stock = setKey(viv0.stock, k, cant))\}$

`asegura` $\{viv.venta = viv0.venta\}$

`asegura` $\{res = viv.caja\}$

`proc actualizar` (inout viv:Vivero, in k:K, precioVenta:$\mathbb{R}$) : $\mathbb{R}$

`requiere` $\{viv = viv0\ \wedge\ k \in viv.stock\}$

`asegura` $\{viv.caja = viv0.caja\ \wedge\ viv.stock = viv0.stock\ \wedge\ viv.venta = setKey(viv0.venta, k, precioVenta)\}$

`asegura` $\{res = viv.caja\}$

`proc vender` (inout viv:Vivero, in k:K) : $\mathbb{R}$

`requiere` $\{viv = viv0\ \wedge\ k \in viv.stock\ \wedge\ k \in viv.venta\ \wedge\ viv.stock[k] > 0\}$

`asegura` $\{viv.venta = viv0.venta\}$

`asegura` $\{viv.caja = viv0.caja + viv0.venta[k]\}$

`asegura` $\{viv.stock = setKey(viv0.stock, k, (viv0.stock[k] - 1)\}$

`asegura` $\{res = viv.caja\}$