Especificacion de TADs

Ejercicio 1

TAD $NumeroRac\{$

obs num: $\mathbb{Z}$

obs den: $\mathbb{Z}$

proc ***nuevoRac*** (***in*** $n, d : \mathbb{Z}$) : $NumeroRac$ $\{$

requiere $\{d \neq 0\}$

asegura $\{res.num = n \wedge res.den = d\}$

$\}$

proc ***suma*** (***inout*** $a : NumeroRac$, ***in*** $b : NumeroRac$) : $NumeroRac$ $\{$

requiere $\{a = A_0\}$

asegura $\{a.num = A_0.num * b.den + A_0.dem * b.num\}$

$\}$

proc ***resta*** (***inout*** $a : NumeroRac$, ***in*** $b : NumeroRac$) : $NumeroRac$ $\{$

requiere $\{a = A_0\}$

asegura $\{a.num = A_0.num * b.den - A_0.dem * b.num\}$

$\}$

proc ***multiplicacion*** (***inout*** $a : NumeroRac$, ***in*** $b : NumeroRac$) : $NumeroRac$ $\{$

requiere $\{a = A_0\}$

asegura $\{a.num = A_0.num * b.num \wedge a.den = A_0.den * b.den\}$

$\}$

proc ***division*** (***inout*** $a : NumeroRac$, ***in*** $b : NumeroRac$) : $NumeroRac$ $\{$

requiere $\{a = A_0\}$

asegura $\{a.num = A_0.num * b.den \wedge a.den = A_0.den * b.num\}$

$\}$

proc ***iguales*** (***in*** $a, b : NumeroRac$) : $Bool$ $\{$

requiere $\{True\}$

asegura $\{res = True \leftrightarrow a.num * b.den = a.dem * b.num\}$

$\}$

Ejercicio 2

TAD $Rectangulo2D\{$

obs vsi: $tupla(\mathbb{R}, \mathbb{R})$

obs vsd: $tupla(\mathbb{R}, \mathbb{R})$

obs vii: $tupla(\mathbb{R}, \mathbb{R})$

obs vid: $tupla(\mathbb{R}, \mathbb{R})$

aux ***resta*** (***in*** $a, b : tupla(\mathbb{R}, \mathbb{R})$) : $tupla(\mathbb{R}, \mathbb{R})$ = $(a_0 - b_0, a_1 - b_1)$

aux ***prod*** (***in*** $a, b : tupla(\mathbb{R}, \mathbb{R})$) : $\mathbb{R}$ = $a_0 * b_o + a_1 * b_1$

pred ***noRepes*** (***in*** $t : seq\langle\mathbb{R}\rangle$) $\{$

$(\forall i, j : \mathbb{Z})(0 \leq i, j < |t| \wedge i \neq j \rightarrow_L t[i] \neq t[j])\}$

```
proc nuevoRectangulo (in a, b, c, d : ℝ) : Rectangulo2D {
    requiere {noRepes(⟨a, b, c, d⟩)}
    asegura {prod(resta(a.vsi, b.vsd), resta(a.vsi, c.vii)) = 0}
    asegura {prod(resta(d.vid, b.vsd), resta(d.vid, c.vii)) = 0}
    asegura {prod(resta(c.vii, a.vsi), resta(c.vii, d.vid)) = 0}
    asegura {prod(resta(b.vsd, a.vsi), resta(b.vsd, d.vid)) = 0}
}

proc mover (inout r : Rectangulo2D, in dx, dy : ℝ) : Rectangulo2D {
    requiere {r = R_0}
    asegura {r.vsi = (R_0.vsi_0 + dx, R_0.vsi_1 + dy)}
    asegura {r.vsi = (R_0.vsd_0 + dx, R_0.vsd_1 + dy)}
    asegura {r.vsi = (R_0.vii_0 + dx, R_0.vii_1 + dy)}
    asegura {r.vsi = (R_0.vid_0 + dx, R_0.vid_1 + dy)}
}
```

Ejercicio 3

a) TAD $Cola\langle T\rangle$ {

```
    obs s:  seq⟨T⟩

    proc nuevaCola (in s : seq⟨T⟩) : cola⟨T⟩ {
        requiere {True}
        asegura {|res.s| = 0}
    }

    proc estaVacia (in c : cola⟨T⟩) : Bool {
        requiere {True}
        asegura {res = True ↔ |c.s| = 0}
    }

    proc encolar (inout c : cola⟨T⟩, e : T) : cola⟨T⟩ {
        requiere {c = C_0}
        asegura {c.s = concat(C_0.s, {e})}
    }

    proc desencolar (inout c : cola⟨T⟩) : T {
        requiere {c = C_0}
        asegura {c.s = tail(C_0.s) ∧ res = head(C_0.s)}
    }
```

b) TAD $Pila\langle T\rangle$ {

```
    obs s:  seq⟨T⟩

    proc nuevaPila (in s : seq⟨T⟩) : pila⟨T⟩ {
        requiere {True}
        asegura {|res.s| = 0}
    }
```

proc *estaVacia* (*in* $c : pila\langle T\rangle$):$Bool$ {
requiere $\{True\}$
asegura $\{res = True \leftrightarrow |c.s| = 0\}$
}

proc *apilar* (*inout* $c : pila\langle T\rangle, e : T$):$pila\langle T\rangle$ {
requiere $\{c = C_0\}$
asegura $\{c.s = concat(\{e\}, C_0.s)\}$
}

proc *desapilar* (*inout* $c : pila\langle T\rangle$):T {
requiere $\{c = C_0\}$
asegura $\{c.s = tail(C_0.s) \wedge res = head(C_0.s)\}$
}

c) TAD $dobleCola\langle T\rangle$\{

obs $elems$: $seq\langle T\rangle$

proc *nuevaDobleCola* ( ):$dobleCola\langle T\rangle$ {
requiere $\{True\}$
asegura $\{|res.elems| = 0\}$
}

proc *estaVacia* (*in* $c : dobleCola\langle T\rangle$):$Bool$ {
requiere $\{True\}$
asegura $\{|res.elems| = 0\}$
}

proc *encolarAdelante* (*inout* $c : dobleCola\langle T\rangle$, *in* $e : T$): {
requiere $\{c = C_0\}$
asegura $\{c.elems = \{e\} + + C_0.elems\}$
}

proc *encolarAtras* (*inout* $c : dobleCola\langle T\rangle$, *in* $e : T$): {
requiere $\{c = C_0\}$
asegura $\{c.elems = C_0.elems + + \{e\}\}$
}

proc *desencolar* (*inout* $c : dobleCola\langle T\rangle$):$T$ {
requiere $\{c = C_0\}$
asegura $\{res = C_0.elems[|C_0.elems|/2] \wedge c.elems =$
$subseq(C_0.elems, 0, |C_0.elems|/2) + + subseq(C_0.elems, |C_0.elems|/2, |C_0.elems|)\}$
}

Ejercicio 4

a) TAD $Diccionario\langle K, V\rangle\{$

obs $elem$: $conj\langle tupla\langle K, V\rangle\rangle$

proc *nuevoDict* ( ):$Diccionario\langle K, V\rangle$ {
asegura $\{|res.pares| = 0\}$
}

proc *definir* (*inout* $d : Diccionario\langle K, V\rangle$, *in* $k : K$, *in* $t : T$): {
requiere $\{d = D_0\}$
asegura $\{(\exists t' : tupla\langle K, V\rangle)(t' \in D_0 \wedge t'_0 = k) \Leftrightarrow d.pares = (D_0.pares - \langle t'\rangle) \cup \langle\langle k, t\rangle\rangle\}$
asegura $\{\neg(\exists t' : tupla\langle K, V\rangle)(t' \in D_0 \wedge t'_0 = k) \Leftrightarrow d.pares = D_0.pares \cup \langle\langle k, t\rangle\rangle\}$
}

proc *obtener* (*in* $d : Diccionario\langle K, V\rangle$, *in* $k : K$):T {
requiere $\{claveInDict(d, k)\}$
asegura $\{\langle k, res\rangle \in d.pares\}$
}

proc *esta* (*in* $d : Diccionario\langle K, V\rangle$, *in* $k : K$):$Bool$ {
requiere $\{True\}$
asegura $\{res = True \Leftrightarrow claveInDict(d, k)\}$
}

proc *borrar* (*inout* $d : Diccionario\langle K, V\rangle$, *in* $k : K$): {
requiere $\{d = D_0 \wedge claveInDict(D_0, k)\}$
asegura $\{(\exists t : T)(\langle k, t\rangle \in D_0.pares) \Leftrightarrow d.pares = D_0.pares - \langle\langle k, t\rangle\rangle\}$
}

pred *claveInDict* (*in* $d : Diccionario\langle K, V\rangle$, *in* $k : K$){
$(\exists t : tupla\langle K, V\rangle)(t \in d.pares \wedge t_0 = k)\}$

Ejercicio 5

b) TAD $conjunto\langle T\rangle\{$

obs $esta(x : T)$: $Bool$

proc *nuevoConj* ( ):$conjunto\langle T\rangle$ {
asegura $\{(\forall x : T)(\neg res.esta(x))\}$
}

proc *agregar* (*inout* $c : conjunto\langle T\rangle$, *in* $e : T$): {
requiere $\{c = C_0\}$
asegura $\{C_0.esta(e) \Leftrightarrow (\forall x : T)(C_0.esta(x) \rightarrow c.esta(x))\}$
asegura $\{\neg C_0.esta(e) \Leftrightarrow (\forall x : T)(C_0.esta(x) \wedge x \neq e \rightarrow c.esta(x) \wedge c.esta(e))\}$
}

proc *eliminar* (*inout* $c : conjunto\langle T\rangle$, *in* $e : T$): {
requiere $\{c = C_0 \land C_0.esta(e)\}$
asegura $\{(\forall x : T)(C_0.esta(x) \land x \neq e \leftrightarrow c.esta(x) \land \neg c.esta(e))\}$
}

proc *intersec* (*in* $a, b : conjunto\langle T\rangle$):$conjunto\langle T\rangle$ {
requiere $\{True\}$
asegura $\{(\forall x : T)(a.esta(x) \land b.esta(x) \leftrightarrow res.esta(x))\}$
}

proc *union* (*in* $a, b : conjunto\langle T\rangle$):$conjunto\langle T\rangle$ {
requiere $\{True\}$
asegura $\{(\forall x : T)(a.esta(x) \lor b.esta(x) \leftrightarrow res.esta(x))\}$
}

d) TAD *punto*{

obs rad: $\mathbb{R}$
obs ang: $\mathbb{R}$

proc *crearPunto* (*in* $r, a\mathbb{R}$):$punto$ {
asegura $\{res.rad = r \land res.ang = a\}$
}

proc *dist* (*in* $p1, p2 : punto$):$\mathbb{R}$ {
requiere $\{True\}$
asegura $\{res = norma(x(p1) - x(p2), y(p1) - y(p2))\}$
}

proc *distOrig* (*in* $p : punto$):$\mathbb{R}$ {
requiere $\{True\}$
asegura $\{norma(x(p), y(p))\}$
}

proc *mover* (*inout* $p : punto$, *in* $dx, dy : \mathbb{R}$): {
requiere $\{p = P_0\}$
asegura $\{(\exists r, a : \mathbb{R})(r * cos(a) = P_0.rad * cos(P_0.ang) + dx \land r * sen(a) = P_0.rad * sen(P_0.ang) + dy) \leftrightarrow p.rad = r \land p.ang = a\}$
}

aux *x* (*in* $p : punto$):$\mathbb{R}$=$cos(p.ang) * p.rad$
aux *y* (*in* $p : punto$):$\mathbb{R}$=$sen(p.ang) * p.rad$
aux *norma* (*in* $x, y : \mathbb{R}$):$\mathbb{R}$=$\sqrt{x^2 + y^2}$

Ejercicio 6

a) TAD $multiConjunto\langle T\rangle\{$

obs $esta(x : T)$: $Bool$
obs $mult(x : T)$: $\mathbb{Z}$

proc *nuevoMultiConjunto* ( ):$multiConjunto\langle T\rangle$ {
asegura $\{(\forall t : T)(\neg res.esta(t) \leftrightarrow res.mult(t) = 0)\}$
}

proc *agregar* (*inout* $c : multiConjunto\langle T\rangle$, *in* $t : T$): {
requiere $\{c = C_0\}$
asegura $\{(\forall x : T)(C_0.esta(x) \land x \neq t \leftrightarrow c.esta(x) \land c.esta(t) \land C_0.mult(x) = c.mult(x) \land$
$c.mult(t) = C_0.mult(t) + 1)\}$
}

proc *quitar* (*inout* $c : multiConjunto\langle T\rangle$, *in* $t : T$): {
requiere $\{c = C_0\}$
asegura $\{C_0.mult(t) > 1 \land (\forall x : T)(C_0.esta(x) \land x \neq t \leftrightarrow c.esta(x) \land c.esta(t) \land$
$C_0.mult(x) = c.mult(x) \land c.mult(t) = C_0.mult(t) - 1)\}$
asegura $\{C_0.mult(t) \leq 1 \land (\forall x : T)(C_0.esta(x) \land x \neq t \leftrightarrow c.esta(x) \land \neg c.esta(t) \land$
$C_0.mult(x) = c.mult(x) \land c.mult(t) = 0)\}$
}

proc *pertenece* (*in* $c : multiConjunto\langle T\rangle$, *in* $t : T$):$Bool$ {
requiere $\{True\}$
asegura $\{res = c.esta(t)\}$
}

proc *multiplicidad* (*in* $c : multiConjunto\langle T\rangle$, *in* $t : T$):$\mathbb{Z}$ {
requiere $\{True\}$
asegura $\{res = c.mult(t)\}$
}

b) TAD $multiDict\langle K, V\rangle\{$

obs $data$: $seq\langle tupla\langle K, V\rangle\rangle$

proc *nuevoMultiDIct* ( ):$multiDict\langle K, V\rangle$ {
asegura $\{|res.data| = 0\}$
}

proc *definir* (*inout* $md : multiDict\langle K, V\rangle$, *in* $k : K$, *in* $t : T$): {
requiere $\{md = Md_0\}$
asegura $\{cantApariciones(\langle K, V\rangle, md.data) = cantApariciones(\langle K, V\rangle, Md_0.data) + 1\}$
}

proc *obtener* (*in* $md : multiDict\langle K, V\rangle$, *in* $k : K$) : $seq\langle V\rangle$ {
    requiere $\{True\}$
    asegura $\{(\forall t : tupla\langle K, V\rangle)(t \in md.data \wedge t_0 = k \leftrightarrow t_1 \in res)\}$
}

proc *pertenece* (*in* $md : multiDict\langle K, V\rangle$, *in* $k : K$) : $Bool$ {
    requiere $\{True\}$
    asegura $\{res = True \leftrightarrow claveInDict(md, k)\}$
}

proc *borrar* (*inout* $md : multiDict\langle K, V\rangle$, *in* $k : K$) : {
    requiere $\{md = Md_0\}$
    asegura $\{(\forall t : tupla\langle K, V\rangle)(t \in Md_0.data \wedge t_0 = k \leftrightarrow \neg(t \in md.data))\}$
}

pred *claveInDict* (*in* $md : multiDict\langle K, V\rangle$, *in* $k : K$){
    $(\exists t : tupla\langle K, V\rangle)(t \in md.data \wedge t_0 = k)\}$
aux *cantApariciones* (*in* $t : T$, *in* $s : seq\langle T\rangle$) : $\mathbb{Z} = \sum_{i=0}^{|s|-1} ifThenElse(s[i] = t, 1, 0)$

Ejercicio 7

i) TAD *contador*{

obs $list$: $seq\langle T\rangle$
obs $cont(x : T)$: $\mathbb{Z}$

proc *nuevoContador* (*in* $s : seq\langle T\rangle$) : *contador* {
    asegura $\{(\forall t : T)(t \in s \rightarrow res.cont(t) = cantApariciones(t, s))\}$
}

proc *cantEventos* (*in* $c : contador$, *in* $t : T$) : $\mathbb{Z}$ {
    requiere $\{True\}$
    asegura $\{res = c.cont(t)\}$
}

proc *incrementarEvento* (*inout* $c : contador$, *in* $t : T$) : {
    requiere $\{t \in c.list \wedge c = C_0\}$
    asegura $\{c.list = C_0.list + +\langle t\rangle \wedge c.cont(t) = C_0.cont(t) + 1\}$
}

ii) TAD *contador*{

obs $list$: $seq\langle T\rangle$
obs $cont(x : T)$: $dict(\mathbb{Z}, \mathbb{Z})$

proc *nuevoContador* (*in* $s : seq\langle T\rangle$, *in* $fecha : \mathbb{Z}$) : *contador* {
    asegura $\{(\forall t : T)(t \in s \leftrightarrow setKey(res.cont(t), fecha, cantApariciones(t, s)))\}$
    asegura $\{(\forall t : T)(t \in s \leftrightarrow t \in c.list)\}$
}

proc *cantEventos* (*in* $c : contador$, *in* $t : T$, *in* $fecha : \mathbb{Z}$) : $\mathbb{Z}$ {
    requiere $\{True\}$
    asegura $\{(\exists f : \mathbb{Z})(f \in c.cont(t) \wedge (\forall f' : \mathbb{Z})(f' \in c.cont(t) \Leftrightarrow f' \leq f \leq fecha)) \leftrightarrow res = d[f]\}$
}

proc *incrementarEvento* (*inout* $c : contador$, *in* $t : T$, *in* $fecha : \mathbb{Z}$) : {
    requiere $\{t \in c.list \wedge c = C_0\}$
    asegura $\{c.list = C_0.list + +\langle t \rangle\}$
    asegura $\{cantApariciones(t, c.list) = cantApariciones(t, C_0.list) + 1\}$
    asegura $\{setKey(c.cont(t), fecha, cantApariciones(t, C_0.list) + 1)\}$
}

proc *eventosPorFecha* (*in* $c : contador$, *in* $t : T$, *in* $fecha : \mathbb{Z}$) : $\mathbb{Z}$ {
    requiere $\{fecha \in c.cont(t)\}$
    asegura $\{res = c.cont(t)[fecha]\}$
}

Ejercicio 8

i) TAD $CacheFIFO\langle K, V \rangle$ {

obs $data$: $dict(K, V)$
obs $time$: $dict(K, \mathbb{Z})$
obs $capacidad$: $\mathbb{Z}$

proc *nuevoCacheFIFO* (*in* $cap : \mathbb{Z}$) : $CacheFIFO\langle K, V \rangle$ {
    requiere $\{cap \geq 0\}$
    asegura $\{res.capacidad = cap\}$
    asegura $\{|res.data| = 0 \wedge |res.time| = 0\}$
}

proc *esta* (*in* $c : CacheFIFO\langle K, V \rangle$, *in* $k : K$) : $Bool$ {
    requiere $\{True\}$
    asegura $\{res = True \leftrightarrow k \in c.data \wedge k \in c.time\}$
}

proc *obtener* (*in* $c : CacheFIFO\langle K, V \rangle$, *in* $k : K$) : V {
    requiere $\{k \in c.data \wedge k \in c.time\}$
    asegura $\{res = c.data[k]\}$
}

proc *definir* (*inout* $c : CacheFIFO\langle K, V \rangle$, *in* $k : K, \in v : V$) : {
    requiere $\{c = C_0\}$
    asegura $\{C_0.cap = 0 \leftrightarrow c.cap = 0 \wedge c.data = \{\} \wedge c.time = \{\}\}$
    asegura $\{C_0.cap = |C_0.data| \wedge k \in C_0.data \leftrightarrow setKey(c.data, k, v) \wedge setKey(c.time, k, horaActual())\}$
    asegura $\{C_0.cap = |C_0.data| \wedge \neg(k \in C_0.data) \leftrightarrow setKey(delKey(c.data, m), k, v) \wedge setKey(delKey(c.time, m), k, horaActual) \wedge claveMasVieja(C_0, m)\}$

asegura $\{C_0.cap > |C_0.data| \leftrightarrow setKey(c.data, k, v) \land setKey(c.time, k, horaActual())\}$
asegura $\{c.cap = C_0.cap\}$
}

pred *claveMasVieja* $(c : CacheFIFO\langle K, V\rangle, k : K)\{$
$(\forall p : K)(p \in c.time \leftrightarrow c.time[p] \geq c.time[k])\}$

ii) TAD $CacheLRU\langle K, V\rangle\{$

obs $data$: $dict(K, V)$
obs $time$: $dict(K, \mathbb{Z})$
obs $access$: $dict(K, \mathbb{Z})$
obs $capacidad$: $\mathbb{Z}$

proc *nuevoCacheLRU* (*in* $cap : \mathbb{Z}$) : $CacheLRU\langle K, V\rangle$ {
requiere $\{cap \geq 0\}$
asegura $\{res.capacidad = cap\}$
asegura $\{|res.data| = 0 \land |res.time| = 0 \land |res.access| = 0\}$
}

proc *esta* (*in* $c : Cache\langle K, V\rangle$, *in* $k : K$) : $Bool$ {
requiere $\{True\}$
asegura $\{res = True \leftrightarrow k \in c.data \land k \in c.time\}$
}

proc *obtener* (*in* $c : Cache\langle K, V\rangle$, *in* $k : K$) : V {
requiere $\{k \in c.data \land k \in c.time\}$
asegura $\{res = c.data[k] \land setKey(c.access, k, horaActual())\}$
}

proc *definir* (*inout* $c : CacheLRU\langle K, V\rangle$, *in* $k : K, \in v : V$): {
requiere $\{c = C_0\}$
asegura $\{C_0.cap = 0 \leftrightarrow c.cap = 0 \land c.data = \{\} \land c.time = \{\} \land c.access = \{\}\}$
asegura $\{C_0.cap = |C_0.data| \land k \in C_0.data \leftrightarrow c.data = setKey(c.data, k, v) \land$
$c.time = setKey(c.time, k, horaActual()) \land (\forall k' : K)(k' \in C_0.access \Leftrightarrow k' \in c.access)\}$
asegura $\{C_0.cap = |C_0.data| \land \neg(k \in C_0.data) \leftrightarrow$
$c.data = setKey(delKey(C_0.data, m), k, v) \land$
$c.time = setKey(delKey(C_0.time, m), k, horaActual) \land$
$((oldestAccess(C_0, m) \land c.access = delKey(C_0.access, m)) \lor$
$\neg oldestAccess(C_0, m) \land claveMasVieja(C_0, m))\}$
asegura $\{C_0.cap > |C_0.data| \leftrightarrow c.data = setKey(C_0.data, k, v) \land$
$c.time = setKey(C_0.time, k, horaActual()) \land (\forall k' : K)(k' \in C_0.access \Leftrightarrow k' \in c.access)\}$
asegura $\{c.cap = C_0.cap\}$
}

pred *oldestKey* $(c : CacheLRU\langle K, V\rangle, k : K)\{$
$(\forall p : K)(p \in c.time \leftrightarrow c.time[p] \geq c.time[k])\}$
pred *oldestAccess* $(c : CacheLRU\langle K, V\rangle, k : K)\{$
$(\forall p : K)(p \in c.access \leftrightarrow c.access[p] \geq c.access[k])\}$

iii) TAD $CacheTTL\langle K, V\rangle\{$

obs $data$: $dict(K, V)$
obs $time$: $dict(K, \mathbb{Z})$
obs $tmax$: $\mathbb{Z}$

proc *nuevoCacheTTL* (*in* $tiempoMax : \mathbb{Z}$):$CacheTTL\langle K, V\rangle$ {
requiere $\{cap \geq 0\}$
asegura $\{res.tmax = tiempoMax\}$
asegura $\{|res.data| = 0 \land |res.time| = 0\}$
}

proc *esta* (*in* $c : CacheTTL\langle K, V\rangle$, *in* $k : K$):$Bool$ {
requiere $\{True\}$
asegura $\{res = True \leftrightarrow k \in c.data \land k \in c.time \land c.time[k] < c.tmax\}$
}

proc *obtener* (*in* $c : CacheTTL\langle K, V\rangle$, *in* $k : K$):V {
requiere $\{k \in c.data \land k \in c.time \land c.time[k] < c.tmax\}$
asegura $\{res = c.data[k]\}$
}

proc *definir* (*inout* $c : CacheLRU\langle K, V\rangle$, *in* $k : K, \in v : V$): {
requiere $\{c = C_0\}$
asegura $\{c.data = setKey(C_0.data, k, v) \land c.time = setKey(C_0.time, k, time)\}$
asegura $\{c.tmax = C_0.tmax\}$
}

Ejercicio 9

TAD $robot\{$

obs pos: $struct\langle x : \mathbb{Z}, y : \mathbb{Z}\rangle$
obs $cont(p : struct\langle x : \mathbb{Z}, y : \mathbb{Z}\rangle)$: $\mathbb{Z}$

proc *nuevoRobot* (*in* $posx, posy : \mathbb{Z}$):$robot$ {
asegura $\{res.pos.x = posx \land res.pos.y = posy\}$
asegura $\{(\forall a, b : \mathbb{Z})(a \neq posx \land b \neq posy \leftrightarrow res.cont(\langle x : a, y : b\rangle) = 0)\}$
asegura $\{res.cont(\langle x : posx, y : posy\rangle) = 1\}$
}

proc *arriba* (*inout* $r : robot$): {
requiere $\{r = R_0\}$
asegura $\{r.pos.x = R_0.pos.x \land r.pos.y = R_0.pos.y + 1\}$
asegura $\{r.cont(\langle x : r.pos.x, y : r.pos.y\rangle) = R_0.cont(\langle x : R_0.pos.x, y : R_0.pos.y\rangle) + 1\}$
}

proc *abajo* (*inout* $r : robot$): {
requiere $\{r = R_0\}$
asegura $\{r.pos.x = R_0.pos.x \land r.pos.y = R_0.pos.y - 1\}$
asegura $\{r.cont(\langle x : r.pos.x, y : r.pos.y\rangle) = R_0.cont(\langle x : R_0.pos.x, y : R_0.pos.y\rangle) + 1\}$
}

proc *derecha* (**inout** $r : robot$): {
requiere $\{r = R_0\}$
asegura $\{r.pos.x = R_0.pos.x + 1 \land r.pos.y = R_0.pos.y\}$
asegura $\{r.cont(\langle x : r.pos.x, y : r.pos.y\rangle) = R_0.cont(\langle x : R_0.pos.x, y : R_0.pos.y\rangle) + 1\}$
}

proc *izquierda* (**inout** $r : robot$): {
requiere $\{r = R_0\}$
asegura $\{r.pos.x = R_0.pos.x - 1 \land r.pos.y = R_0.pos.y\}$
asegura $\{r.cont(\langle x : r.pos.x, y : r.pos.y\rangle) = R_0.cont(\langle x : R_0.pos.x, y : R_0.pos.y\rangle) + 1\}$
}

proc *masDerecha* (**in** $r : robot$): $\langle x : \mathbb{Z}, y : \mathbb{Z}\rangle$ {
requiere $\{True\}$
asegura $\{res = c \leftrightarrow res.cont(c) > 0 \land (\forall c' : \langle x : \mathbb{Z}, y : \mathbb{Z}\rangle)(res.cont(c') > 0 \leftrightarrow c'.x \leq c.x)\}$
}

proc *cuantasVecesPaso* (**in** $r : robot$, **in** $t : \langle x : \mathbb{Z}, y : \mathbb{Z}\rangle$): $\mathbb{Z}$ {
requiere $\{True\}$
asegura $\{res = r.cont(t)\}$
}

Ejercicio 10

TAD *vivero*{
obs *stock*: $dict(K, \mathbb{Z})$
obs *precios*: $dict(K, \mathbb{R})$
obs *balance*: $\mathbb{R}$

proc *nuevoVivero* (**in** *capital*): $Vivero$ {
asegura $\{|res.data| = 0 \land |res.stock| = 0 \land res.balance = capital\}$
}

proc *comprar* (**inout** $v : vivero$, **in** $planta : string$, **in** $cant : \mathbb{Z}$, **in** $precio : \mathbb{R}$): {
requiere $\{capital \geq cant * precio \land cant > 0 \land precio > 0\}$
requiere $\{v = V_0\}$
asegura $\{planta \in v.stock \leftrightarrow v.stock = setKey(V_0.stock, planta, V_0.stock[planta] + cant)\}$
asegura $\{planta \notin v.stock \leftrightarrow v.stock = setKey(V_0.stock, planta, cant)\}$
asegura $\{v.balance = V_0.balance - cant * precio\}$
asegura $\{v.precios = V_0.precios\}$
}

proc *asignarPrecio* (**inout** $v : vivero$, **in** $planta : string$, **in** $p : precio$): {
requiere $\{v = V_0 \land planta \in V_0.stock \land p > 0\}$
asegura $\{v.precios = setKey(V_0.stock, planta, p) \land v.stock = V_0.stock \land$
$v.balance = V_0.balance\}$
}

proc *venta* (**inout** $v : vivero$, **in** $planta : string$): {

requiere $\{V_0.stock[planta] > 0 \land planta \in V_0.precios \land v = V_0\}$

asegura $\{v.stock[planta] = V_0.stock[planta] - 1 \land v.precios = V_0.precios\}$

asegura $\{v.balance = V_0.balance + v.precios[planta]\}$

}

proc balance (in $v : vivero$):$\mathbb{R}$ {

requiere $\{True\}$

asegura $\{res = v.balance\}$

}

Ejercicio 11

TAD $dobleCola\langle T\rangle${

obs $cola1$: $seq\langle T\rangle$

obs $cola2$: $seq\langle T\rangle$

obs $lastCola$: $\mathbb{Z}$

proc nuevaDobleCola (in $c1, c2 : seq\langle T\rangle$):$dobleCola\langle T\rangle$ {

requiere $\{noHayRepetidos(c1 + +c2)\}$

asegura $\{res.cola1 = c1 \land res.cola2 = c2 \land lastCola = 2\}$

}

proc encolar (inout $dc : dobleCola\langle T\rangle$ in $elem : T$, in $cola : \mathbb{Z}$): {

requiere $\{dc = Dc_0 \land (cola = 1 \lor cola = 2)\}$

asegura $\{cola = 1 \leftrightarrow dc.cola1 = Dc_0.cola1 + +\{elem\} \land dc.cola2 = Dc_0.cola2\}$

asegura $\{cola = 2 \leftrightarrow dc.cola2 = Dc_0.cola2 + +\{elem\} \land dc.cola1 = Dc_0.cola1\}$

asegura $\{dc.lastCola = Dc_0.lastCola\}$

}

proc desencolar (inout $dc : dobleCola\langle T\rangle$):$T$ {

requiere $\{dc = Dc_0 \land (|Dc_0.cola1| > 0 \lor |Dc_0.cola2| > 0)\}$

asegura $\{Dc_0.lastCola = 2 \land |Dc_0.cola1| > 0 \leftrightarrow$
$dc.cola1 = tail(Dc_0.cola1) \land res = head(Dc_0.cola1) \land dc.lastCola = 1\}$

asegura $\{Dc_0.lastCola = 1 \land |Dc_0.cola2| > 0 \leftrightarrow$
$dc.cola2 = tail(Dc_0.cola2) \land res = head(Dc_0.cola2) \land dc.lastCola = 2\}$

asegura $\{Dc_0.lastCola = 2 \land |Dc_0.cola1| = 0 \land |Dc_0.cola2| > 0 \leftrightarrow$
$dc.cola2 = tail(Dc_0.cola2) \land res = head(Dc_0.cola2) \land dc.lastCola = 1\}$

asegura $\{Dc_0.lastCola = 1 \land |Dc_0.cola2| = 0 \land |Dc_0.cola1| > 0 \leftrightarrow$
$dc.cola1 = tail(Dc_0.cola1) \land res = head(Dc_0.cola1) \land dc.lastCola = 2\}$

}

proc mudarElemento (inout $dc : dobleCola\langle T\rangle$, in $elem : T$): {

requiere $\{elem \in (Dc_0.cola1 + +Dc_0.cola2) \land dc = Dc_0\}$

asegura $\{esIndice(i, elem, Dc_0.cola1) \leftrightarrow$
$dc.cola1 = subseq(Dc_0.cola1, 0, i) + +subseq(Dc_0.cola1, i + 1, |Dc_0.cola1|) \land$
$dc.cola2 = Dc_0.cola2 + +\{elem\}\}$

asegura $\{esIndice(i, elem, Dc_0.cola2) \leftrightarrow$
$dc.cola2 = subseq(Dc_0.cola1, 0, i) + +subseq(Dc_0.cola2, i + 1, |Dc_0.cola2|) \land$
$dc.cola1 = Dc_0.cola1 + +\{elem\}\}$

}

pred *noHayRepetidos* ($s : seq\langle T\rangle$){
$(\forall i, j : T)(i \in s \land j \in s \rightarrow i \neq j)$}
pred *esIndice* ($i : \mathbb{Z}, elem : T, s : seq\langle T\rangle$){
$res = True \leftrightarrow 0 \leq i < |s| \land_L s[i] = elem$}

Ejercicio 12

TAD $CallOfDuty : The\ Arrival\ of\ AED$\{
obs $batallones$: $dict(K, seq\langle\mathbb{Z}\rangle)$
obs $money$: $\mathbb{R}$
obs $lands$: $seq\langle T\rangle$
Nota: por comodidad se usara la equivalencia cod=CallOfDuty:The Arrival of AED

proc *CoDInit* (*in* $m : \mathbb{R}$, *in* $land : K$, *in*):cod {
requiere $\{m \geq 0\}$
asegura $\{res.batallones = \{\} \land res.money = m \land res.lands = land\}$
}

proc *contratarMercenario* (*inout* $c : cod$, *in* $bat : K$, *in* $mer : struct\langle power : \mathbb{Z}, prize : \mathbb{R}\rangle$):
{
requiere $\{c = C_0 \land C_0.money \geq mer.prize\}$
requiere $\{mer.prize \geq 0 \land mer.power \geq 0\}$
asegura $\{|C_0.batallones[bat]|| > 0 \leftrightarrow$
$c.batallones = setKey(C_0.batallones, bat, C_o.batallones[bat] + +\{mer.power\})\}$
asegura $\{|C_0.batallones[bat]|| = 0 \leftrightarrow c.batallones = setKey(C_0.batallones, bat, \{mer.power\})\}$
asegura $\{(\forall k : K)(k \in C_0.batallones \land k \neq bat \leftrightarrow c.batallones[k] = C_0.batallones[k])\}$
asegura $\{c.money = C_0.money - mer.prize\}$
asegura $\{c.lands = C_0.lands\}$
}

proc *atacarTerritorio* (*inout* $c : cod$, *in* $bat : k$, *in* $terr : T$, *in* $power : \mathbb{Z}$): {
requiere $\{c = C_0\}$
requiere $\{terr \notin C_0.lands \land power \geq 0 \land C_0.money \geq 500\}$
asegura $\{poderBat(C_0, bat) > power \leftrightarrow c.money = C_0 - money + 1000 \land$
$c.lands = C_0.lands + +\{terr\}\}$
asegura $\{poderBat(C_0, bat) \leq power \leftrightarrow c.money = C_0 - money - 500 \land c.lands = C_0.lands\}$
asegura $\{c.batallones = C_0.batallones\}$
}

proc *cantTerritorios* (*in* $c : cod$):$\mathbb{Z}$ {
asegura $\{res = |c.lands|\}$
}

aux *poderBat* (*in* $c : cod$, *in* $bat : K$):$\mathbb{Z}$=$\sum_{i=0}^{|c.batallones[bat]|-1} c.batallones[bat][i]$