Algoritmos y Estructura de Datos 2

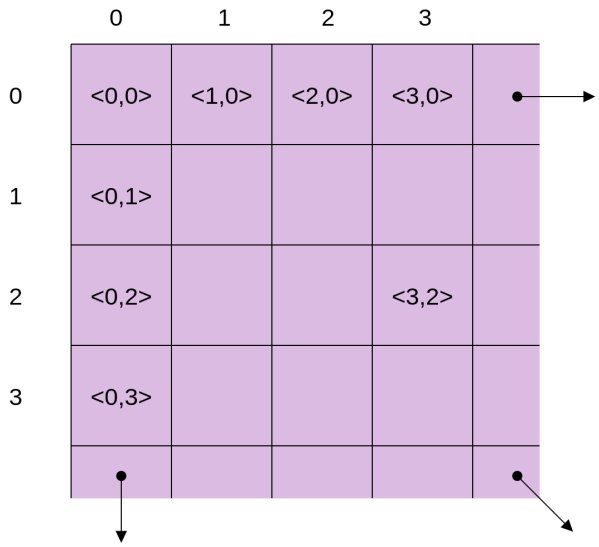
Trabajo Práctico 2

Especificación de *Sokoban Extendido*

Alumno: Leandro Carreira
LU: 669/18
Grupo: 15

Documento online: docs.google.com/document/d/...

Disposición de grilla



TAD Posición **es** tupla(nat, nat)
TAD Personaje **es** Posición
TAD Caja **es** Posición
TAD Depósito **es** Posición
TAD Pared **es** Posición
TAD Bombas **es** Nat

TAD Acción **es** Enum { Arriba, Abajo, Izquierda, Derecha, Explotar }
TAD Estado **es** tupla(Personaje, Bombas, conj(Caja), conj(Depósito), conj(Pared))

TAD Mapa(Personaje, Bombas, Cajas, Depósitos, Paredes)

géneros mapa

exporta observadores, generadores, nivelVálido?, completado?

usa Bool, Nat, Conjunto, Posición, Personaje, Caja, Depósito, Pared, Acción, Bombas

igualdad observacional

$$
\begin{aligned}
&(\forall m1, m2 : \textbf{mapa}) \\
&\quad (m1 =_{obs} m2 \iff (\\
&\qquad \textbf{(}\ personaje(m1) =_{obs} personaje(m2) \quad \wedge \\
&\qquad\quad bombas(m1) =_{obs} bombas(m2) \quad \wedge \\
&\qquad\quad cajas(m1) =_{obs} cajas(m2) \quad \wedge \\
&\qquad\quad depósitos(m1) =_{obs} depósitos(m2) \quad \wedge \\
&\qquad\quad paredes(m1) =_{obs} paredes(m2) \quad \wedge \\
&\qquad\quad historial(m1) =_{obs} historial(m2) \\
&\qquad \textbf{)} \\
&\quad))
\end{aligned}
$$

observadores básicos

personaje	: mapa	→	personaje
bombas	: mapa	→	nat
cajas	: mapa	→	conj(caja)
depósitos	: mapa	→	conj(depósito)
paredes	: mapa	→	conj(pared)
historial	: mapa	→	secu(Estado)

generadores

crearMapa	: Personaje pe × Nat × Cajas ca × Depósitos de × Paredes pa	→	Mapa	{ nivelVálido?(pe, ca, de, pa) }
moverArriba	: mapa m	→	mapa	{ acciónEjecutable?(m,Arriba) }
moverAbajo	: mapa m	→	mapa	{ acciónEjecutable?(m,Abajo) }

```
moverIzquierda    : mapa m            → mapa    { acciónEjecutable?(m,Izquierda) }
moverDerecha      : mapa m            → mapa    { acciónEjecutable?(m,Derecha) }
tirarBomba        : mapa m            → mapa    { acciónEjecutable?(m,Explotar) }
retroceder        : mapa m × nat n    → mapa          { n ≤ long(historial(m)) }
```

otras operaciones

```
agregarAHistorial : Mapa × Secu(Estado)    → Secu(Estado)
obtenerElemDeSecu : nat  × secu(α)         → α
acciónEjecutable? : mapa × tupla(nat, nat) → bool
moverCaja         : Posición × conj(Caja) × tupla(nat, nat) → conj(Caja)
borrarCruz        : Posición × Conj(Pared)                  → conj(Pared)
borrarÚltimos     : secu(α)  × nat                          → secu(α)


nivelVálido?      : Personaje × conj(Caja) × conj(Depósito) × conj(Pared) → bool
esPared?          : conj(Pared) × Posición   → bool
esCaja?           : conj(Caja)  × Posición   → bool
estáLibre?        : Mapa × Posición          → bool
completado?       : Mapa                     → bool

sumar2Tuplas      : tupla(nat, nat) × tupla(nat, nat) → tupla(nat, nat)
sumar3Tuplas      : tupla(nat, nat) × tupla(nat, nat) × tupla(nat, nat)
                                                    → tupla(nat, nat)
```

axiomas

```
personaje(crearMapa(pe, n, ca, de, pa)) ≡ pe
personaje(moverArriba(m))    ≡ sumar2Tuplas( personaje(m), ⟨0,-1⟩ )
personaje(moverAbajo(m))     ≡ sumar2Tuplas( personaje(m), ⟨0, 1⟩ )
personaje(moverIzquierda(m)) ≡ sumar2Tuplas( personaje(m), ⟨-1,0⟩ )
personaje(moverDerecha(m))   ≡ sumar2Tuplas( personaje(m), ⟨ 1,0⟩ )
personaje(tirarBomba(m))     ≡ personaje(m)
personaje(retroceder(m, n))  ≡
            π1(obtenerElemDeSecu(long(historial(m)) - n, historial(m)))

historial(crearMapa(pe, n, ca, de, pa)) ≡  ⟨ pe, n, ca, de, pa ⟩
historial(moverArriba(m))    ≡ agregarAHistorial(moverArriba(m), historial(m))
historial(moverAbajo(m))     ≡ agregarAHistorial(moverAbajo(m), historial(m))
historial(moverIzquierda(m)) ≡ agregarAHistorial(moverIzquierda(m), historial(m))
historial(moverDerecha(m))   ≡ agregarAHistorial(moverDerecha(m), historial(m))
historial(tirarBomba(m))     ≡ agregarAHistorial(tirarBomba(m), historial(m))
historial(retroceder(m, r))  ≡ borrarÚltimos( historial(m), r )

cajas(crearMapa(pe, n, ca, de, pa)) ≡ ca
cajas(moverArriba(m))    ≡ moverCaja( sumar2Tuplas(personaje(m),  ⟨0,-1⟩ ),
                                      cajas(m), ⟨0,-1⟩ )
cajas(moverAbajo(m))     ≡ moverCaja( sumar2Tuplas(personaje(m),  ⟨0, 1⟩ ),
                                      cajas(m), ⟨0, 1⟩ )
```

```
cajas(moverIzquierda(m)) ≡ moverCaja( sumar2Tuplas(personaje(m), ⟨-1, 0⟩ ),
                                      cajas(m), ⟨-1, 0⟩ )
cajas(moverDerecha(m))   ≡ moverCaja( sumar2Tuplas(personaje(m), ⟨1, 0⟩ ),
                                      cajas(m), ⟨1, 0⟩      )
cajas(tirarBomba(m))     ≡ cajas(m)

cajas(retroceder(m, n))  ≡
           π₃(obtenerElemDeSecu(long(historial(m)) - n, historial(m)))

depósitos(crearMapa(pe, n, ca, de, pa)) ≡ de
depósitos(moverArriba(m))     ≡ depósitos(m)
depósitos(moverAbajo(m))      ≡ depósitos(m)
depósitos(moverIzquierda(m))  ≡ depósitos(m)
depósitos(moverDerecha(m))    ≡ depósitos(m)
depósitos(tirarBomba(m))      ≡ depósitos(m)
depósitos(retroceder(m, n))   ≡ depósitos(m)

paredes(crearMapa(pe, n, ca, de, pa)) ≡ pa
paredes(moverArriba(m))       ≡ paredes(m)
paredes(moverAbajo(m))        ≡ paredes(m)
paredes(moverIzquierda(m))    ≡ paredes(m)
paredes(moverDerecha(m))      ≡ paredes(m)
paredes(retroceder(m, n))     ≡ paredes(m)
paredes(tirarBomba(m))        ≡ borrarCruz(personaje(m), paredes(m))

borrarCruz : Posición × Conj(Pared) → conj(Pared)
borrarCruz(pos, paredes) ≡
           if vacío?(paredes) then
                 paredes
           else
                 if π₁(pos) = π₁(dameUno(paredes)) ∨
                    π₂(pos) = π₂(dameUno(paredes))
                 then
                      borrarCruz(pos, sinUno(paredes))
                 else
                      Ag(dameUno(paredes) , borrarCruz(pos, sinUno(paredes)))
                 fi
           fi

bombas(crearMapa(pe, n, ca, de, pa)) ≡ n
bombas(moverArriba(m))     ≡ bombas(m)
bombas(moverAbajo(m))      ≡ bombas(m)
bombas(moverIzquierda(m))  ≡ bombas(m)
bombas(moverDerecha(m))    ≡ bombas(m)
bombas(retroceder(m, n))   ≡ bombas(m)
bombas(tirarBomba(m))      ≡ bombas(m) - 1
bombas(retroceder(m, n)) ≡
```

```
            π₂(obtenerElemDeSecu(long(historial(m)) - n, historial(m)))

agregarAHistorial : Mapa × Secu(Estado) → Secu(Estado)
agregarAHistorial(m, h) ≡
                ⟨ personaje(m),
                    bombas(m),
                     cajas(m),
                  depósitos(m),
                    paredes(m) ⟩ • h

obtenerElemDeSecu : nat x secu(α) → α
obtenerElemDeSecu(n, s) ≡
            if n = 0 then
                  prim(s)
            else
                  obtenerElemDeSecu(n-1, fin(s))
            fi

borrarÚltimos: secu(α) × nat → secu(α)
borrarÚltimos(s, n) ≡
            if n = 0 ∨ vacía?(s) then
                  s
            else
                  borrarÚltimos(fin(s), n-1)
            fi

acciónEjecutable? : mapa × tupla(nat, nat) → bool
acciónEjecutable?(m, paso) ≡
                  estáLibre?(m, sumar2Tuplas(personaje(m), paso))          ∨
                  ( esCaja?( cajas(m), sumar2Tuplas(personaje(m), paso) )  ∧
                    estáLibre?( m, sumar3Tuplas(personaje(m), paso, paso) )  )

moverCaja : Posición × conj(Caja) × tupla(nat, nat) → conj(Caja)
moverCaja(posCaja, cajas, paso) ≡
    if dameUno(cajas) = pos then
        Ag(sumar2Tuplas(posCaja, paso), sinUno(cajas))
    else
        Ag(dameUno(cajas), moverCaja(posCaja, sinUno(cajas), paso))
    fi

nivelVálido? : Personaje × conj(Caja) × conj(Depósito) × conj(Pared) → bool
nivelVálido?(personaje, cajas, depósitos, paredes) ≡
                  -- requerimientos del enunciado
                  ¬(personaje ∈ cajas)        ∧
                  ¬(personaje ∈ paredes)      ∧
                  #(cajas) = #(depósitos)     ∧
                  vacío?(paredes ∩ cajas)     ∧
```

vacío?(paredes ∩ depósitos**)**

```
esPared? : conj(Pared) × Posición → bool
esPared?(c, p) ≡
            if Ø?(c) then
                  false
            else
                  if dameUno(c) = p then
                        true
                  else
                        esPared?(sinUno(c), p)
                  fi
            fi
```

```
esCaja? : conj(Caja) × Posición → bool
esCaja?(c, a) ≡
            if Ø?(c) then
                   false
            else
                  if dameUno(c) = a then
                        true
                  else
                        esCaja?(sinUno(c), a)
                  fi
            fi
```

```
estáLibre? : Mapa × Posición → bool
estáLibre?(m, p) ≡ ¬esPared?(paredes(m), p) ∧ ¬esCaja?(cajas(m), p)
```

```
completado? : Mapa          → bool
completado?(m) ≡ paredes(m) = depósitos(m)
```

```
sumar2Tuplas : tupla(nat, nat) × tupla(nat, nat) →  tupla(nat, nat)
sumar2Tuplas(a, b) ≡ < π1(a) + π1(b) , π2(a) + π2(b) >
```

```
sumar3Tuplas : tupla(nat, nat) × tupla(nat, nat) × tupla(nat, nat)
                                                  →  tupla(nat, nat)
sumar3Tuplas(a, b, c) ≡ < π1(a) + π1(b) + π1(c) , π2(a) + π2(b) + π2(c) >
```

Fin TAD

TAD Juego(conj(Mapa))

géneros juego

exporta observadores, generadores

usa Bool, Nat, Acción, Mapa, Conjunto(Mapa)

igualdad observacional

$$(\forall j1, j2 : \textbf{juego}) \; (j1 =_{obs} j2 \iff ((\; mapaActual(j1) =_{obs} mapaActual(j2) \; \wedge \; completados(j1) =_{obs} completados(j2) \; \wedge \; incompletos(j1) =_{obs} incompletos(j2) \;) \;) \;)$$

observadores básicos

mapaActual : juego → mapa
completados : juego → conj(mapa)
incompletos : juego → conj(mapa)

generadores

iniciar : conj(Mapa) cm → juego { nivelesVálidos?(cm) }
accionar : juego j × Acción → juego { ¬completado?(mapaActual(j)) }
undo : juego j × Nat → juego { n ≤ long(historial(mapaActual(j))) }

otras operaciones

computarMapa : Juego × Mapa → Mapa
todosLosMapas : Juego → conj(Mapa)
computarMapasCompletados : Juego × Mapa → conj(Mapa)
computarMapasIncompletos : Juego × Mapa → conj(Mapa)

nivelesVálidos? : conj(Mapa) → bool

axiomas

mapaActual(iniciar(cm)) ≡ **dameUno(**cm**)** -- comienza en algún nivel
mapaActual(accionar(j, a)) ≡
-- uso generadores de TAD Mapa
if a = Arriba **then**
computarMapaActual(j, moverArriba(mapaActual(j)))
else if a = Abajo **then**
computarMapaActual(j, moverAbajo(mapaActual(j)))
else if a = Izquierda **then**
computarMapaActual(j, moverIzquierda(mapaActual(j)))
else if a = Derecha **then**
computarMapaActual(j, moverDerecha(mapaActual(j)))
else if a = Explotar **then**
computarMapaActual(j, tirarBomba(mapaActual(j)))

```
            else fi fi fi fi fi
mapaActual(undo(j, n)) ≡ retroceder(mapaActual(j), n)


computarMapaActual(juego, mapa) ≡
            if completado?(mapa) then
                  if ¬∅?(incompletos(juego)) then
                        dameUno(incompletos(juego))
                  else
                        -- Devuelvo alguno al azar, pues ganó todos
                        -- NO puede seguir jugando por la Pre de accionar
                        dameUno(todosLosMapas(juego))
                  fi
            else
                  mapa
            fi

todosLosMapas(j) ≡ Ag(mapaActual(j), completados(j) U incompletos(j))

completados(iniciar(cm))    ≡ 0
completados(accionar(j, a)) ≡
            -- uso generadores de TAD Mapa
            if a = Arriba then
                  computarMapasCompletados(j, moverArriba(mapaActual(j)))
            else if a = Abajo then
                  computarMapasCompletados(j, moverAbajo(mapaActual(j)))
            else if a = Izquierda then
                  computarMapasCompletados(j, moverIzquierda(mapaActual(j)))
            else if a = Derecha then
                  computarMapasCompletados(j, moverDerecha(mapaActual(j)))
            else if a = Explotar then
                  -- no se puede completar un mapa por explotar una bomba
                  completados(j)
            else fi fi fi fi fi
completados(undo(j, n)) ≡ completados(j)

computarMapasCompletados(juego, mapa) ≡
            if completado?(mapa) then
                  Ag(mapa, completados(juego))
            else
                  completados(juego)
            fi

incompletos(iniciar(cm))    ≡ cm
incompletos(accionar(j, a)) ≡
            -- uso generadores de TAD Mapa
            if a = Arriba then
```

```
                        computarMapasIncompletos(j, moverArriba(mapaActual(j)))
                  else if a = Abajo then
                        computarMapasIncompletos(j, moverAbajo(mapaActual(j)))
                  else if a = Izquierda then
                        computarMapasIncompletos(j, moverIzquierda(mapaActual(j)))
                  else if a = Derecha then
                        computarMapasIncompletos(j, moverDerecha(mapaActual(j)))
                  else if a = Explotar then
                        -- no se puede completar un mapa por explotar una bomba
                        incompletos(j)
                  else fi fi fi fi fi
      incompletos(undo(j, n)) ≡ incompletos(j)

      computarMapasIncompletos(juego, mapa) ≡
                  if completado?(mapa) then
                        mapa - incompletos(juego)
                  else
                        incompletos(juego)
                  fi

      nivelesVálidos?(c) ≡ #(c) > 0 ∧
                  if #(c) = 1 then
                        nivelVálido?(dameUno(c)) -- uso de Mapa
                  else
                        nivelVálido?(dameUno(c)) ∧ nivelesVálidos?(sinUno(c))
                  fi
Fin TAD
```
