DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

TP de Especificación

Juego de la vida toroidal

4 de octubre de 2018 — Algoritmos y Estructuras de Datos I

Grupo: FrankerZ

Integrante	LU	Correo electrónico
Manuel Panichelli	072/18	panicmanu@gmail.com
Ignacio Alonso Rehor	195/18	arehor.ignacio@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

1. Problemas

type $toroide = seq\langle seq\langle \mathsf{Bool} \rangle\rangle$

1.1. esValido

```
proc esValido (in t: toroide, out result: Bool) {
        Pre {true}
        Post {result = true ↔ esToroideValido(t)}
}
```

1.2. posicionesVivas

```
proc posicionesVivas (in t: toroide, out vivas: seq⟨ℤ × ℤ⟩) {
        Pre {esToroideValido(t)}
        Post {(sonPosicionesValidas(vivas, t) ∧_L
                sonPosicionesVivas(vivas, t)) ∧
               (cantidadDePosicionesVivas(t) = |vivas|)}
}
   pred sonPosicionesValidas (ps: seq⟨ℤ × ℤ⟩, t: toroide) {
        (∀ p : ℤ × ℤ) (p ∈ ps ⟶ esPosicionValida(t, p))
   }

   pred sonPosicionesVivas (vivas: seq⟨ℤ × ℤ⟩, t: toroide) {
        (∀ p : ℤ × ℤ) (p ∈ vivas ⟶_L esPosicionViva(t, p))
   }
```

1.3. densidadPoblacion

```
proc densidadPoblacion (in t: toroide, out result: ℝ) {
        Pre {esToroideValido(t)}
        Post {result × cantidadTotalDePosiciones(t) = cantidadDePosicionesVivas(t)}
}
   aux cantidadTotalDePosiciones (t: toroide) : ℤ = cols(t) × rows(t) ;
```

1.4. evolucionDePosicion

```
proc evolucionDePosicion (in t: toroide, in p: ℤ × ℤ , out result: Bool) {
        Pre {esToroideValido(t) ∧ esPosicionValida(t, p)}
        Post {result = true ↔ viveLuegoDeUnTick(t, p)}
}
```

1.5. evolucionToroide

```
proc evolucionToroide (inout t: toroide) {
        Pre {esToroideValido(t) ∧ t = T_0}
        Post {esToroideValido(t) ∧ esEvolucion(t, T_0)}
}
```

1.6. evolucionMultiple

```
proc evolucionMultiple (in t : toroide, in k : ℤ, out result : toroide)  {
        Pre {esToroideValido(t) ∧ (k ≥ 1)}
        Post {esK-EsimaEvolucion(result, t, k)}
}
```

1.7. esPeriodico

```
proc esPeriodico (in t : toroide, inout p: ℤ, out result : Bool)  {
        Pre {esToroideValido(t)}
        Post {(result = true ↔ tienePeriodicidad(t)) ∧
              esMinimaPeriodicidad(t, p)}
}
```

1.8. primosLejanos

```
proc primosLejanos (in t1 : toroide, in t2 : toroide, out primos : Bool)  {
        Pre {esToroideValido(t1) ∧ esToroideValido(t2)}
        Post {primos = true ↔ sonPrimosLejanos(t1, t2)}
}

/* Chequeamos que k sea mayor o igual a cero porque consideramos
   que un toroide es primo lejano de si mismo.                    */

   pred sonPrimosLejanos (t1: toroide, t2: toroide) {
        (∃ k : ℤ)((k ≥ 0) ∧_L (esK-EsimaEvolucion(t1, t2, k) ∨ esK-EsimaEvolucion(t2, t1, k)))
   }
```

1.9. seleccionNatural

```
proc seleccionNatural (in ts: seq⟨toroide⟩, out res: ℤ)  {
        Pre {sonToroidesValidos(ts)}
        Post {(0 ≤ res < |ts|) ∧_L
              (noMuere(ts[res]) ∨ esElDeMuerteMasTardia(ts[res], ts))}
}
   pred noMuere (t: toroide) {
        tienePeriodicidad(t)
   }

   pred esElDeMuerteMasTardia (t: toroide, ts: seq⟨toroide⟩) {
        (∀ w : toroide) (w ∈ ts ⟶_L muereDespues(t, w))
   }

   pred muereDespues (t, w: toroide) {
        (∃ n, m : ℤ) ((n, m ≥ 0) ∧_L (muereEnTick(t, n) ∧ muereEnTick(w, m) ∧ (n ≥ m)))
   }

/* Un toroide t muere en k ticks si en su k-esima evolución esta muerto
   y en todas las anteriores esta vivo                                 */
```

```
pred muereEnTick (t: toroide, k: ℤ) {
    (∃ t_k : toroide) (esToroideValido(t_k) ∧_L
        (esK-EsimaEvolucion(t_k, t, k) ∧ estaMuerto(t_k) ∧
        lasEvolucionesAnterioresEstanVivas(t, k)))
}

pred lasEvolucionesAnterioresEstanVivas (t: toroide, k: ℤ) {
    (∀ q : ℤ) ((0 ≤ q < k) ⟶_L
        ((∃ t_q : toroide) (esToroideValido(t_q) ∧_L
            (esK-EsimaEvolucion(t_q, t, q) ∧ ¬ estaMuerto(t_q)))))
}

pred estaMuerto (t: toroide) {
    (∀ p : ℤ × ℤ) (esPosicionValida(t, p) ⟶_L ¬ esPosicionViva(t, p))
}
```

1.10. fusionar

```
proc fusionar (in t1: toroide, in t2: toroide, out res: toroide) {
    Pre {esToroideValido(t1) ∧
         esToroideValido(t2) ∧
         mismaDimension(t1, t2)}
    Post {(esToroideValido(res) ∧
          mismaDimension(t1, res)) ∧_L
          compartenPosicionesVivas(t1, t2, res)}
}

/* Supone que t1, t2, y tr tienen la misma dimension. */

pred compartenPosicionesVivas (t1, t2, res: toroide) {
    (∀ p : ℤ × ℤ) (esPosicionValida(t1, p) ⟶_L
        (esPosicionViva(res, p) ↔ (esPosicionViva(t1, p) ∧ esPosicionViva(t2, p))))
}
```

1.11. vistaTrasladada

```
proc vistaTrasladada (in t1: toroide, in t2: toroide, out res: Bool) {
    Pre {esToroideValido(t1) ∧
         esToroideValido(t2) ∧
         mismaDimension(t1, t2)}
    Post {res = true ↔ esTraslacion(t1,t2)}
}
```

1.12. enCrecimiento

```
proc enCrecimiento (in t: toroide, out res: Bool) {
    Pre {esToroideValido(t)}
    Post {res = true ↔ areaQueCubrePosVivasIncrementa(t)}
}

/* Dado un toroide queremos ver todas las matrices validas
```

```
   que encierran las posiciones vivas de ese toroide y
   quedarnos con la de area mínima.
   Asi mismo, vemos las traslaciones pues puede haber una
   vista trasladada de un toroide cuya area que encierra
   a las posiciones vivas sea del menor a la del original. */

   pred areaQueCubrePosVivasIncrementa (t: toroide) {
        (∃ t_1 : toroide) ((esToroideValido(t_1) ∧_L esEvolucion(t_1, t)) ∧
        (∃ a, a_1 : ℤ)
             (esAreaValida(t, a) ∧ esAreaValida(t, a_1) ∧
              esMinimaAreaQueCubrePosVivas(t, a) ∧
              esMinimaAreaQueCubrePosVivas(t_1, a_1) ∧
              a < a_1))
   }

   pred esAreaValida (t: toroide, a: ℤ) {
        0 ≤ a ≤ area(t)
   }

   pred esMinimaAreaQueCubrePosVivas (t: toroide, a: ℤ) {
        esAreaQueCubrePosVivas(t, a) ∧
        (∀ b : ℤ) ((esAreaValida(t, b) ∧ (esAreaQueCubrePosVivas(t, b)) ⟶ a ≤ b)
   }

   pred esAreaQueCubrePosVivas (t: toroide, a: ℤ) {
        (∃ t_d : toroide) ((esToroideValido(t_d) ∧_L esTraslacion(t_d, t)) ∧
             ((∃ m : seq⟨seq⟨T⟩⟩)
                  (esMatriz(m) ∧
                  (area(m) = a) ∧
                  cubrePosicionesVivas(t_d, m))))
   }

   aux area (m: seq⟨seq⟨T⟩⟩) : ℤ = cols(m) × rows(m) ;

/* Queremos ver que si una posicion está viva en el toroide,
   esté contenida en la matriz que mide el area.
   Para esto necesitamos poder comparar las coordenadas del
   toroide con las de la matriz.
   El problema es que sus coordenadas no son las mismas, sino
   que pueden tener un desfazaje con respecto al toroide.

   Este será la posicion del eje de coordenadas de la
   matriz con respecto al del toroide.

   Por ejemplo, un toroide que tenga una sola posición
   viva en la esquina inferior derecha, tendrá una matriz que
   la contenga con un desfazaje de (cols(t) - 1, 0)              */

   pred cubrePosicionesVivas (t: toroide, m: seq⟨seq⟨T⟩⟩) {
        (∃ e : ℤ × ℤ) (esDesfazajeValido(e, m, t) ∧_L
             ((∀ p : ℤ × ℤ) (esPosicionValida(t, p) ∧_L esPosicionViva(t, p)) ⟶_L
                  estaDentro(m, posicionRelativa(p, e))))
   }

   pred esDesfazajeValido (e: ℤ × ℤ , m: seq⟨seq⟨T⟩⟩, t: toroide) {
```

```
    (0 ≤ r0 + cols(m) ≤ cols(t)) ∧
    (0 ≤ r1 + rows(m) ≤ rows(t))
}

pred estaDentro (m: seq⟨seq⟨T⟩⟩, p: ℤ × ℤ ) {
    (0 ≤ p0 < cols(m)) ∧
    (0 ≤ p1 < rows (m))
}

aux posicionRelativa (p: ℤ × ℤ , r: ℤ × ℤ ) : ℤ × ℤ =
    (p0 - r0, p1 - r1) ;
```

2. Predicados y Auxiliares generales

2.1. Toroides

```
pred esToroideValido (t: toroide) {
    esMatriz(t) ∧ (rows(t) ≥ 3) ∧ (cols(t) ≥ 3)
}

pred sonToroidesValidos (ts: seq⟨toroide⟩) {
    (∀ t : toroide) (t ∈ ts ⟶ esToroideValido(t))
}

/* Supone que es una posicion válida (no se va a ir de rango) */

aux valorEn (t: toroide, p: ℤ × ℤ ) : Bool = t[p0][p1] ;
pred esPosicionViva (t: toroide, p: ℤ × ℤ ) {
    (valorEn(t, p) = true)
}

aux cantidadDePosicionesVivas (t: toroide) : ℤ =
    cantidadDeAparicionesEnMat(true, t) ;

pred esEvolucion (t, t0: toroide) {
    mismaDimension(t, t0) ∧L
    (∀ p : ℤ × ℤ) (esPosicionValida(t, p) ⟶L valorEn(t, p) = viveLuegoDeUnTick(t0, p))
}

pred mismaDimension (t, t0: toroide) {
    (cols(t) = cols(t0)) ∧
    (rows(t) = rows(t0))
}

pred esK-EsimaEvolucion (t, t0: toroide, k: ℤ) {
    (∃ ts : seq⟨toroide⟩)
        ((|ts| > k) ∧L sonToroidesValidos(ts) ∧L
            ((ts[0] = t0) ∧ (ts[k] = t) ∧
            (∀ i : ℤ) ((0 ≤ i < |ts| - 1) ⟶L esEvolucion(ts[i + 1], ts[i])))))
}

pred tienePeriodicidad (t: toroide) {
    (∃ k : ℤ)(tienePeriodicidadK-Esima(t, k))
}
```

```
pred esMinimaPeriodicidad (t: toroide, p: ℤ) {
    tienePeriodicidadK-Esima(t, p) ∧
    (∀ q : ℤ) (tienePeriodicidadK-Esima(t, q) ⟶ p ≤ q)
}

pred tienePeriodicidadK-Esima (t: toroide, k: ℤ) {
    (k ≥ 1) ∧ (esK-EsimaEvolucion(t, t, k))
}

aux trasladar (t: toroide, p: ℤ × ℤ , d: ℤ × ℤ ) : ℤ × ℤ =
    (mod(p_0 + d_0, cols(t),
     mod(p_1 + d_1, rows(t)) ;

/* Para traslaciones suponemos que son todos de la misma
   dimension                                              */

pred esTraslacion (t1, t2: toroide) {
    (∃ d : ℤ × ℤ) esTraslacionEnDireccion(t1, t2, d)
}

pred esTraslacionEnDireccion (t, t_0: toroide, d: ℤ × ℤ ) {
    (∀ p : ℤ × ℤ) (esPosicionValida(t_0, p) ⟶_L
        (valorEn(t_0, p) = valorEn(t, trasladar(t_0, p, d))))
}
```

2.1.1. Vecinos

```
pred viveLuegoDeUnTick (t: toroide, p: ℤ × ℤ ) {
    (¬ esSoledad(t, p) ∧ ¬ esSuperpoblacion(t, p)) ∧
    (esSupervivencia(t, p) ∨ esReproduccion(t, p))
}
pred esSoledad (t: toroide, p: ℤ × ℤ ) {
    esPosicionViva(t, p) ∧ (cantidadVecinosVivos(t, p) < 2)
}
pred esSuperpoblacion (t: toroide, p: ℤ × ℤ ) {
    esPosicionViva(t, p) ∧ (cantidadVecinosVivos(t, p) > 3)
}
pred esSupervivencia (t: toroide, p: ℤ × ℤ ) {
    esPosicionViva(t, p) ∧ (2 ≤ cantidadVecinosVivos(t, p) ≤ 3)
}
pred esReproduccion (t: toroide, p: ℤ × ℤ ) {
    ¬ esPosicionViva(t, p) ∧ (cantidadVecinosVivos(t, p) = 3)
}
aux cantidadVecinosVivos (t: toroide, p: ℤ × ℤ ) : ℤ × ℤ =
    ∑_{i=-1}^{1} ∑_{j=-1}^{1} if (¬ esPosicionViva(t, trasladar(t, p, (i, j)) ∨ (i, j) = (0,0))
                              then 0
                              else 1 fi ;
```

2.2. Matrices

```
/* Para rows y cols suponemos que m es una matriz */

aux cols (m: seq⟨seq⟨T⟩⟩) : ℤ = |m| ;
```

```
aux rows (m: seq⟨seq⟨T⟩⟩) : ℤ = if cols(m) > 0 then |m[0]| else 0 fi ;
pred esMatriz (m: seq⟨seq⟨T⟩⟩) {
    (∀ i, j : ℤ) ((0 ≤ i, j < |m|) ⟶_L |m[i]| = |m[j]|)
}

aux cantidadDeAparicionesEnMat (x: T, m: seq⟨seq⟨T⟩⟩) : ℤ =
    ∑_{i=0}^{cols(m)-1} ∑_{j=0}^{rows(m)-1} (if s[i][j] = x then 1 else 0 fi) ;

pred esPosicionValida (m: seq⟨seq⟨T⟩⟩, p: ℤ × ℤ ) {
    (0 ≤ p_0 < cols(m)) ∧
    (0 ≤ p_1 < rows(m))
}
```

3. Decisiones tomadas

- Tomamos al Toroide como una Matriz compuesta por una $seq\langle seq\langle \mathsf{Bool}\rangle\rangle$ Donde la primera representa a las columnas, y la segunda a las filas. Accederemos a ella mediante tuplas $(x, y) : \mathbb{Z}\times\mathbb{Z}$, donde x es la posición en la columna e y en la fila.

- Tomamos como decisión que un toroide válido será aquel que sea al menos de 3×3, de forma tal que los 8 vecinos de cualquier posicion sean siempre diferentes entre sí.

- Asumimos que *mod* es una función dada, que cumple con la siguiente especificación:

```
proc mod (in n: ℤ, in m: ℤ, out result: ℤ) {
        Pre {m > 0}
        Post {(result ≥ 0) ∧ ((∃ q : ℤ) n = q × m + result)}
}
```