DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TPI - Informe

10 de junio de 2016 — Algoritmos y Estructuras de Datos I

Grupo 13

Integrante	LU	Correo electrónico
Seijo, Jonathan Adrián	592/15	jon.seijo@gmail.com
De Bortoli, Lucas Gabriel	736/15	lu_cas_.97@hotmail.com.ar
Córdoba, Lucas Mauricio	094/15	lmcordobaa@gmail.com
Galli, Luciano Martín	534/15	lucianogalli@outlook.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. Informe

1.1. Campo

- En el constructor vacío de Campo creamos una Grilla de tres parcelas: un Granero, una Casa y un Cultivo, lo cual cumple el invariante del tipo Campo.

1.2. Drone

- En el constructor vacío de Drone ponemos 'posicionActual' en (0,0), 'batería' en 100, 'enVuelo' false y 'trayectoria' la dejamos vacía, lo cual cumple el invariante del tipo.
- En el constructor con parámetros de Drone, y teniendo en cuenta que no toma un Posición para 'posicionActual' decidimos ubicarlo en el (0,0), lo cual cumple el invariante del tipo.
- En 'vueloEscalerado' decidimos, por simplicidad para la demostración, hacer explícito el hecho de que toda trayectoria de dos posiciones o menos es escalerada.

1.3. Sistema

- En el contructor vacío de Sistema decidimos, por declaratividad, crear explícitamente un Campo de 3 Parcelas y una lista vacía de drones y asignarlos a las correspondientes variables de Sistema.
- En volarYSensar, si el Drone llega a una Parcela en estado 'NoSensado', decidimos cambiarla a 'RecienSembrado', ya que la esecificación solo dice que deja de ser NoSensado.

2. Vuelo Escalerado

```
problema _escalerado (ps: [Posicion]) = res : bool {
        asegura (∃x, y ← [1, −1])(∀i ← [0..|ps| − 2))(prm(ps[i]) − prm(ps[i + 2]) == x ∨ sgd(ps[i]) − sgd(ps[i + 2]) == y);
}

problema _esEscalerado (ps: [Posicion], x, y: ℤ) = res : bool {
        requiere |ps| >= 2;
        asegura (∀i ← [0..|ps| − 2))(prm(ps[i]) − prm(ps[i + 2]) == x ∨ sgd(ps[i]) − sgd(ps[i + 2]) == y);
}
```

```
bool Drone::vueloEscalerado() const{
    return _enVuelo && _escalerado();
     // Estado E1:
     // Vale: res == enVuelo(this) &&
     // (∃ x, y ← [1, −1])
     // (∀ i ← [0..|ps| − 2))
     // (prm(ps[i]) - prm(ps[i+2]) == x ||
     // sgd(ps[i]) - sgd(ps[i+2]) == y )
}

bool Drone::_escalerado(const std::vector<Posicion> ps) const{
    return (ps.size() ≤ 2) || _esEscalerado(ps,1,1) || _esEscalerado(ps,1,−1) ||
            _esEscalerado(ps,−1,1) || _esEscalerado(ps,−1,−1);
     // Estado E1:

     // Vale: res == (|ps| ≥ 2) || (∃ x, y ← [1, −1])
     // (∀ i ← [0..|ps| − 2))
     // (prm(ps[i]) - prm(ps[i+2]) == x ||
     // sgd(ps[i]) - sgd(ps[i+2]) == y )
}

bool Drone::_esEscalerado(const std::vector<Posicion> ps, int X, int Y) const{

    bool cumplenTodos = true;
     // Estado E1:
     // Vale: cumplenTodos == true


    unsigned int i = 0;
     // Estado E2:
     // Vale: i == 0 && cumplenTodos == cumplenTodos@E1
    while (i < ps.size()−2){

     // I: (0 ≤ i ≤ |vueloRealizado(this)| - 2) &&
     // (cumplenTodos == (∀ j ← [0..i)) ((prm(vueloRealizado(this)[j]) - (prm(vueloRealizado(this)[j+2]) == X) &&
     // ((sgd(vueloRealizado(this)[j]) - (sgd(vueloRealizado(this)[j+2]) == Y)))

     // B: i < |ps| − 2
     // Fv:|ps| − 2 − i Cota: 0

        if ((ps.at(i).x − ps.at(i+2).x) != X ||
                (ps.at(i).y − ps.at(i+2).y) != Y ){
            cumplenTodos = false;
        }
         // Estado C1:
         // Vale: ( prm(ps[i]) - prm(ps[i+2])) != X ||
         // (sgd(ps[i]) − sgd(ps[i + 2])) != Y && cumplenTodos == false ||
         //(prm(ps[i]) − prm(ps[i + 2])) = X &&
         // (sgd(ps[i]) − sgd(ps[i + 2])) = Y && cumplenTodos == true
        i++;
         // Estado C2:
         // Vale: i == i@C1 + 1 && I
```

```
    }

    // Estado E3:
    // Vale: cumplenTodos == ( ∀ i ← [0..—ps—-2) )
    // (prm(ps[i]) - prm(ps[i+2]) == x ||
    // sgd(ps[i]) - sgd(ps[i+2]) == y)

    return cumplenTodos;
    // Estado E4:
    // Vale: res ==( ∀ i ← [0..—ps—-2) )
    // (prm(ps[i]) - prm(ps[i+2]) == x ||
    // sgd(ps[i]) - sgd(ps[i+2]) == y)
}
```

Demostracion de ciclo: _esEscalerado.

// Vale PC: (i == 0) && (cumplenTodos == True) && (|vueloRealizado(this)| ≥ 2)
// I: (0 ≤ i ≤ |vueloRealizado(this)| - 2) &&
(cumplenTodos == (∀ j ← [0..i)) ((prm($vueloRealizado(this)_{[j]}$) - (prm($vueloRealizado(this)_{[j+2]}$) == X) &&
((sgd($vueloRealizado(this)_{[j]}$) - (sgd($vueloRealizado(this)_{[j+2]}$) == Y)))
// B: i < |vueloRealizado(this)| - 2
// Fv: |vueloRealizado(this)| - 2 - i
// Cota: 0
// QC: cumplenTodos == (∀ j ← [0..|vueloRealizado(this)| - 2))
((prm($vueloRealizado(this)_{[j]}$) - (prm($vueloRealizado(this)_{[j+2]}$) == X) &&
((sgd($vueloRealizado(this)_{[j]}$) - (sgd($vueloRealizado(this)_{[j+2]}$) == Y))

– Pc ⟶ I
i == 0 ⟶ 0 ≤ i ≤ 0
|vueloRealizado(**this**)| ≥ 2 ⟶ |vueloRealizado(**this**)| – 2 ≥ 0

0 ≤ i ≤ 0 && |vueloRealizado(**this**)| – 2 ≥ 0 ⟶
0 ≤ i ≤ |vueloRealizado(**this**)| – 2

i == 0 ⟶ (∀ j ← [0..i)) ((prm($vueloRealizado(\textbf{this})_{[j]}$) – (prm($vueloRealizado(\textbf{this})_{[j+2]}$) == X)
&& ((sgd($vueloRealizado(\textbf{this})_{[j]}$) – (sgd($vueloRealizado(\textbf{this})_{[j+2]}$) == Y)) == True

Reemplazando i==0, queda (∀ j ← [0..0)) (Verdadero por vacuidad del selector [0..0))

cumplenTodos == True &&
((∀ j ← [0..i)) ((prm($vueloRealizado(\textbf{this})_{[j]}$) – (prm($vueloRealizado(\textbf{this})_{[j+2]}$) == X) &&
((sgd($vueloRealizado(\textbf{this})_{[j]}$) – (sgd($vueloRealizado(\textbf{this})_{[j+2]}$) == Y)) == True)
⟶ cumplenTodos == ((∀ j ← [0..i))
((prm($vueloRealizado(\textbf{this})_{[j]}$) – (prm($vueloRealizado(\textbf{this})_{[j+2]}$) == X) &&
((sgd($vueloRealizado(\textbf{this})_{[j]}$) – (sgd($vueloRealizado(\textbf{this})_{[j+2]}$) == Y)) == True)

Vale por transitividad

– I && noB ⟶ Qc

0 ≤ i ≤ |vueloRealizado(**this**)| – 2 && (i ≥ |vueloRealizado(**this**)| – 2) &&
cumplenTodos == (∀ j ← [0..i))
((prm($vueloRealizado(\textbf{this})_{[j]}$) – (prm($vueloRealizado(\textbf{this})_{[j+2]}$) == X) &&
((sgd($vueloRealizado(\textbf{this})_{[j]}$) – (sgd($vueloRealizado(\textbf{this})_{[j+2]}$) == Y))

Estos son el invariante y la negacion de la guarda.

0 ≤ i ≤ |vueloRealizado(**this**)| – 2 && i ≥ |vueloRealizado(**this**)| – 2

$\longrightarrow$ i == |vueloRealizado(**this**)| - 2

i == |vueloRealizado(**this**)| - 2 && ($\forall$ j $\leftarrow$ [0..i))
((prm(vueloRealizado(**this**)$_{[j]}$) - (prm(vueloRealizado(**this**)$_{[j+2]}$) == X) &&
((sgd(vueloRealizado(**this**)$_{[j]}$) - (sgd(vueloRealizado(**this**)$_{[j+2]}$) == Y)))
$\longrightarrow$
($\forall$ j $\leftarrow$ [0..—vueloRealizado(this)— - 2))
((prm(vueloRealizado(**this**)$_{[j]}$) - (prm(vueloRealizado(**this**)$_{[j+2]}$) == X) &&
((sgd(vueloRealizado(**this**)$_{[j]}$) - (sgd(vueloRealizado(**this**)$_{[j+2]}$) == Y)))

- I && Fv < cota $\longrightarrow$ noB

0 $\leq$ i $\leq$ |vueloRealizado(**this**)| - 2 && |vueloRealizado(**this**)| −2 − i) < 0 &&
cumplenTodos == ($\forall$ j $\leftarrow$ [0..i))
((prm(vueloRealizado(**this**)$_{[j]}$) - (prm(vueloRealizado(**this**)$_{[j+2]}$) == X) &&
((sgd(vueloRealizado(**this**)$_{[j]}$) - (sgd(vueloRealizado(**this**)$_{[j+2]}$) == Y))

Estos son el invariante y la funcion variante menor a la cota

|vueloRealizado(**this**)| - 2 - i < 0 $\longrightarrow$ |vueloRealizado(**this**)| - 2 < i
$\longrightarrow$ |vueloRealizado(**this**)| - 2 $\leq$ i

3. Listo Para Cosechar

problema `listoParaCosechar` (this: Sistema) = `res` : bool {
    `asegura` $res == (cantCultivosCosechables(campo(this))/|parcelasCultivo(campo(this))| \geq 0,9)$;
    `aux cantCultivosCosechables` (s: Sistema) : $\mathbb{Z} = |[1|pos \leftarrow parcelasCultivo(campo(s)), estadoDelCultivo(pos, s) == ListoParaCosechar]|$;
    `aux parcelasCultivo` (c: Campo) : $[(\mathbb{Z}, \mathbb{Z})] = [(i,j)|i \leftarrow [0..prm(dimensiones(c))), j \leftarrow [0..sgd(dimensiones(c))), contenido((i,j),c) == Cultivo]$;
}

problema `_cuentaCosechables` (this: Sistema) = `res` : $\mathbb{Z}$ {
    `asegura` $res == cantCultivosCosechables(campo(this))$;
}

problema `_parcelasCultivo` (this: Sistema) = `res` : $\mathbb{Z}$ {
    `asegura` $res == parcelasCultivo(campo(this))$;
}

problema `_parcelasCultivoLargo` (this: Sistema, ($\mathbb{Z}$,$\mathbb{Z}$): parcelasCultivo, $\mathbb{Z}$: fila) = `res` : $\mathbb{Z}$ {
    `modifica` $parcelasCultivo$;
    `asegura` $parcelasCultivo == [(fila, j)|j \leftarrow [0..largo(dimensiones(campo(this)))), contenido(campo(this), fila, j) == Cultivo]$;
}

```
Transformaciones de estados.

bool Sistema::listoParaCosechar() const{

    float cantCosechable = _cuentaCosechables();

    // Estado E1:
    // Vale: cantCosechable == cantCultivosCosechables(campo(this))

    float totalCultivos = _parcelasCultivo().size();

    // Estado E2:
    // Vale: totalCultivos == |parcelasCultivo(campo(this))| && cantCosechable == cantCosechable@E1

    return cantCosechable/totalCultivos >= 0.9f;

    // Estado E3:
    // Vale: res == (cantCosechable@E2 / totalCultivos@E2 ¿= 0,9)
}

int Sistema::_cuentaCosechables() const{
    int total = 0;

    // Estado A1:
    // Vale: total == 0

    unsigned int i = 0;

    // Estado A2:
    // Vale: i == 0 && total == total@A1
    // Vale PC: i == 0 && total == 0

    while(i < _parcelasCultivo().size()){

    // I: 0 ≤ i ≤ |parcelasCultivo(campo(this))| &&
    // total == |[1—a ← [0..i),
          estadoDelCultivo(parcelasCultivo[a], this) == ListoParaCosechar]|
    // B: i < |parcelasCultivo(campo(this))|
    // Fv:|parcelasCultivo(campo(this))|-i Cota: 0
```

```
    // Estado AC1
    // Vale Pif: I && B

        if (estadoDelCultivo(_parcelasCultivo().at(i)) == ListoParaCosechar){
            total++;
        }

    // Estado AC2
    // Vale Qif: total ==|[1|p ← parcelasCultivo,
    estadoDelCultivo(p, this) == ListoParaCosechar]|&& i == i@AC2

        i++;

    // Estado AC3
    // Vale: i == i@AC3+1 && total == total@AC2 && I

    }

    // Estado A3
    // Vale QC: total == cantCultivosCosechables(this)
    return total;

    // Estado A4
    // Vale res == total@A3

}

std::vector<Posicion> Sistema::_parcelasCultivo() const{
    std::vector<Posicion> listaParcelasCultivo;

    // Estado B1
    // Vale: listaParcelasCultivo == []
    int i = 0;
    // Estado B2
    // Vale PC: i == 0 && listaParcelasCultivo == listaParcelasCultivo@B1

    while (i < campo().dimensiones().ancho){
        // I: 0≤i≤|ancho(dimensiones(campo(this)))|&&
        listaParcelasCultivo = [(x,y)|x ← ([0..i)),
                    y ← ([0..largo(dimensiones(campo(this))),
                    contenido(campo(this), x, y) == Cultivo]
        // B: i < ancho(dimensiones(campo(this)))
        // Fv:ancho(dimensiones(campo(this)))-i.Cota:0

        // Estado BC1
        // Vale I && B

        _parcelasCultivoLargo(listaParcelasCultivo, i);

    // Estado BC2
    //Vale: listaParcelasCultivo = [(x,y)|x ← ([0..i]),
        y ← ([0..largo(dimensiones(campo(this))),
        contenido(campo(this), x, y) == Cultivo]
        && i == i@BC1

        i++;

        // Estado BC3
        // Vale: i == i@BC2+1 && I
    }

    // Estado B3
```

```
    // Vale QC: listaParcelasCultivo == parcelasCultivo(campo(this))

    return listaParcelasCultivo;

    // Estado B4
    // Vale res == listaParcelasCultivo@B3

}

void Sistema::_parcelasCultivoLargo(std::vector<Posicion> \&parcelasCultivoLargo,
const int fila) const{
    // Modifica parcelasCultivoLargo
    // Estado C1
    // Vale parcelasCultivoLargo == pre(parcelasCultivoLargo)

    int j = 0;

    // Estado C2
    // Vale PC: j == 0 && parcelasCultivoLargo == parcelasCultivoLargo@C1

    while (j < campo().dimensiones().largo){

        // Estado CC1
        // I: 0 ≤ j ≤ largo(dimensiones(campo(this))) &&
        // parcelasCultivoLargo == pre(parcelasCultivoLargo) ++[(fila,y)|y ← [0..j),
        //contenido(campo(this), fila, y) == Cultivo]
        // B: j < largo(dimensiones(campo(this)))
        // Fv: largo(dimensiones(campo(this)))-j. Cota: 0

        Posicion p;
        p.x = fila;
        p.y = j;

        // Estado CC2
        // Vale Pif: p == (fila, j)

        if (campo().contenido(p) == Cultivo){
            parcelasCultivoLargo.push_back(p);
        }
        // Estado CC3
        // Vale Qif: parcelasCultivoLargo == pre(parcelasCultivoLargo) ++ [(fila,y)|y ← [0..j],
        // contenido(campo(this), fila, y) == Cultivo]

        j++;
        // Estado CC4
        // Vale j == j@CC3+1 && I
    }

    // Estado C3
    // Vale QC: parcelasCultivoLargo == pre(parcelasCultivoLargo) ++[(fila,y)|y ← [0..largo(dimensiones(campo(this))),
            //contenido(campo(this), fila, y) == Cultivo]
}
```

```
Demostracion de ciclo: _cuentaCosechables.

// Vale PC: i == 0 && total == 0
// I: 0 ≤ i ≤ |parcelasCultivo(campo(this))|
// total == |[1—a ← [0..i),
estadoDelCultivo(parcelasCultivo[a], this) == ListoParaCosechar]|
// B: i < |parcelasCultivo(campo(this))|
// Fv:|parcelasCultivo(campo(this))|-i Cota: 0
// QC: total == cantCultivosCosechables(this)




  -  Pc  ⟶  I

i == 0 ⟶ 0 == i ⟶ 0 ≤ i ⟶ 0 ≤ i ≤ 0
⟶ 0 ≤ i ≤ |parcelasCultivo(campo(this))|

total == 0 ⟶ total == |[]| ⟶ total == |[1|a ← [0..0)]|
⟶ total == |[1|a ← [0..i)]|
⟶ total == |[1|a ← [0..i),
     estadoDelCultivo(parcelasCultivo[a], this) == ListoParaCosechar]|


  - I && noB ⟶ Qc

i ≤ |parcelasCultivo(campo(this))| &&  i ≥ |parcelasCultivo(campo(this))|
(invariante y negacion de guarda)
⟶ i == |parcelasCultivo(campo(this))|

total == |[1|\selector{a}{[0..i)},
     estadoDelCultivo(parcelasCultivo[a], this) == ListoParaCosechar]|
⟶ total == |[1|a ← [0..|parcelasCultivo(campo(this))|),
     estadoDelCultivo(parcelasCultivo[a], this) == ListoParaCosechar]|
(pues i == |parcelasCultivo(campo(this))|)
⟶ total == cantCultivosCosechables(this)  (Reemplazo sintactico)


  - (I && Fv < cota) ⟶ noB

|parcelasCultivo(campo(this))|-i < 0     (Fv < cota)
⟶  |parcelasCultivo(campo(this))| < i
⟶  |parcelasCultivo(campo(this))| ≤ i
⟶  noB


_____________________________________________________________________________
_____________________________________________________________________________


Demostracion de ciclo _parcelasCultivo.

     // Pc: i == 0 && listaParcelasCultivo == []
          // I: 0 ≤ i ≤ |ancho(dimensiones(campo(this)))|&&
          listaParcelasCultivo = [(x,y)|x ← ([0..i)),
                         y ← ([0..largo(dimensiones(campo(this))),
                         contenido(campo(this), x, y) == Cultivo]
     // B: i < ancho(dimensiones(campo(this)))
     // Fv: ancho(dimensiones(campo(this)))-i.Cota:0
     // Qc: listaParcelasCultivo == parcelasCultivo(campo(this))
```

```
    - Pc ⟶ I

i == 0 ⟶ 0 == i ⟶ 0 ≤ i ≤ 0
⟶ 0 ≤ i ≤ |[]| ⟶ 0 ≤ i ≤ ancho(dimensiones(campo(this)))

listaParcelasCultivo == [] ⟶ listaParcelasCultivo = [(x,y)|x ← ([0..0)),
                    y ← ([0..largo(dimensiones(campo(this))),
                    contenido(campo(this), x, y) == Cultivo]
    (verdadero por vacuidad del selector [0..0)  )

⟶ listaParcelasCultivo = [(x,y)|x ← ([0..i)),
                    y ← ([0..largo(dimensiones(campo(this))),
                    contenido(campo(this), x, y) == Cultivo]
    (Verdadero pues i == 0)


   - I && noB ⟶ Qc

i≤ancho(dimensiones(campo(this))) &&  i≥ancho(dimensiones(campo(this)))
        (Invariante y NoGuarda)
⟶ i == ancho(dimensiones(campo(this)))

listaParcelasCultivo = [(x,y)|x ← ([0..i)),
                    y ← ([0..largo(dimensiones(campo(this))),
                    contenido(campo(this), x, y) == Cultivo]

    (Verdadero por Invariante)

⟶ listaParcelasCultivo = [(x,y)|x ← ([0..ancho(dimensiones(campo(this)))]),
                    y ← ([0..largo(dimensiones(campo(this))),
                    contenido(campo(this), x, y) == Cultivo]

        (Vale pues i == ancho(dimensiones(campo(this))))


⟶ listaParcelasCultivo == parcelasCultivo(campo(this))  ⟶  Qc
        (Reemplazo sintactico)


    - (I && Fv < cota) ⟶ noB

ancho(dimensiones(campo(this)))-i < 0
⟶  ancho(dimensiones(campo(this))) < i
⟶  i > ancho(dimensiones(campo(this)))
⟶  i ≥  ancho(dimensiones(campo(this))) ⟶  noB
```

Demostracion del ciclo _parcelasCultivoLargo

```
        // Pc: j == 0 && parcelasCultivoLargo == pre(parcelasCultivoLargo)
        // I: 0≤j≤largo(dimensiones(campo(this))) &&
        // parcelasCultivoLargo == pre(parcelasCultivoLargo) ++[(fila,y)|y ← [0..j),
    contenido(campo(this), fila, y) == Cultivo]
    // B: j < largo(dimensiones(campo(this)))
        // Fv: largo(dimensiones(campo(this)))-j. Cota: 0
    // Qc: parcelasCultivoLargo == pre(parcelasCultivoLargo) ++ [(fila,y)|y ← [0..largo(dimensiones(campo(this))),
    // contenido(campo(this), fila, y) == Cultivo]
```

```
    - Pc ⟶ I

j == 0 ⟶ 0 == j ⟶ 0 ≤ j ≤ 0
⟶ 0 ≤ j ≤ |[]| ⟶ 0 ≤ j ≤ largo(dimensiones(campo(this)))

parcelasCultivoLargo == pre(parcelasCultivoLargo) ⟶
parcelasCultivoLargo == pre(parcelasCultivoLargo) ++ []
⟶ parcelasCultivoLargo == pre(parcelasCultivoLargo) ++
                [(fila,y)|\selector{y}{[0..0)}, contenido(campo(this), fila, y) == Cultivo]
(Verdadero porque es lista vacia [0..0) )

⟶ parcelasCultivoLargo == pre(parcelasCultivoLargo) ++
                [(fila,y)|\selector{y}{[0..j)}, contenido(campo(this), fila, y) == Cultivo]
(Porque j == 0)

    - I && noB ⟶ Qc

j≤largo(dimensiones(campo(this))) && j≥largo(dimensiones(campo(this)))
        (Vale por invariante y noB)

⟶ j == largo(dimensiones(campo(this)))

parcelasCultivoLargo == pre(parcelasCultivoLargo) ++
    [(fila,y)|\selector{y}{[0..j)}, contenido(campo(this), fila, y) == Cultivo]

   (Vale por invariante)

⟶ parcelasCultivoLargo == pre(parcelasCultivoLargo) ++
    [(fila,y)|\selector{y}{[0..largo(dimensiones(campo(this)))]},
                contenido(campo(this), fila, y) == Cultivo]

        (Porque j == largo(dimensiones(campo(this)))  )

⟶ Qc


    - (I && Fv < cota) ⟶ noB

largo(dimensiones(campo(this)))-j < 0
(Fv < cota)

⟶ largo(dimensiones(campo(this))) < j

⟶ largo(dimensiones(campo(this))) ≤ j ⟶ noB
```