



# 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 'vueloEscalero' decidimos, por simplicidad para la demostración, hacer explícito el hecho de que toda trayectoria de dos posiciones o menos es escalera.

## 1.3. Sistema

- En el constructor 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 especificación solo dice que deja de ser NoSensado.

## 1.4. Test

- En el test de 'despegar', por una consulta con un ayudante, al mover el Drone por fuera del enjambre su posición real no se modificaba, sacamos esa línea y agregamos la condición de que la posición que ocuparía sea válida. En el paquete agregamos la carpeta test con la modificación aquí descrita

## 2. Vuelo Escaleroado

```

problema _escaleroado (ps: [Posicion]) = res : bool {
    asegura ( $\exists x, y \leftarrow [1, -1]$ ) ( $\forall i \leftarrow [0..|ps| - 2]$ ) ( $prm(ps[i]) - prm(ps[i+2]) == x \vee sgd(ps[i]) - sgd(ps[i+2]) == y$ );
}

problema _esEscaleroado (ps: [Posicion], x, y:  $\mathbb{Z}$ ) = res : bool {
    requiere  $|ps| \geq 2$ ;
    asegura ( $\forall i \leftarrow [0..|ps| - 2]$ ) ( $prm(ps[i]) - prm(ps[i+2]) == x \vee sgd(ps[i]) - sgd(ps[i+2]) == y$ );
}

bool Drone::vueloEscaleroado() const {
    return _enVuelo && _escaleroado();
    // Estado E1:
    // Vale: res == enVuelo(this) &&
    // ( $\exists x, y \leftarrow [1, -1]$ )
    // ( $\forall i \leftarrow [0..|ps| - 2]$ )
    // ( $prm(ps[i]) - prm(ps[i+2]) == x$  ||
    //  $sgd(ps[i]) - sgd(ps[i+2]) == y$ )
}

bool Drone::_escaleroado(const std::vector<Posicion> ps) const {
    return (ps.size()  $\leq$  2) || _esEscaleroado(ps, 1, 1) || _esEscaleroado(ps, 1, -1) ||
        _esEscaleroado(ps, -1, 1) || _esEscaleroado(ps, -1, -1);
    // Estado E1:
    // Vale: res == ( $|ps| \geq 2$ ) || ( $\exists x, y \leftarrow [1, -1]$ )
    // ( $\forall i \leftarrow [0..|ps| - 2]$ )
    // ( $prm(ps[i]) - prm(ps[i+2]) == x$  ||
    //  $sgd(ps[i]) - sgd(ps[i+2]) == y$ )
}

bool Drone::_esEscaleroado(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: ( $0 \leq i \leq |vueloRealizado(this)| - 2$ ) &&
        // ( $cumplenTodos == (\forall j \leftarrow [0..i]) ((prm(vueloRealizado(this)[j]) - (prm(vueloRealizado(this)[j+2]) == X) \&\&$ 
        //  $((sgd(vueloRealizado(this)[j]) - (sgd(vueloRealizado(this)[j+2]) == Y)))$ )
        // 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 == (  $\forall i \leftarrow [0..ps-2]$  )
// (prm(ps[i]) - prm(ps[i+2]) == x ||
// sgd(ps[i]) - sgd(ps[i+2]) == y)

return cumplenTodos;
// Estado E4:
// Vale: res == (  $\forall i \leftarrow [0..ps-2]$  )
// (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(this)[j]) - (prm(vueloRealizado(this)[j+2]) == X)
&& ((sgd(vueloRealizado(this)[j]) - (sgd(vueloRealizado(this)[j+2]) == Y)) == True

```

Reemplazando `i==0`, queda  $(\forall j \leftarrow [0..0])$  (Verdadero por vacuidad del selector  $[0..0]$  )

```

cumplenTodos == True &&
((∀ j ← [0..i]) ((prm(vueloRealizado(this)[j]) - (prm(vueloRealizado(this)[j+2]) == X) &&
((sgd(vueloRealizado(this)[j]) - (sgd(vueloRealizado(this)[j+2]) == Y)) == True)
→ cumplenTodos == ((∀ j ← [0..i])
((prm(vueloRealizado(this)[j]) - (prm(vueloRealizado(this)[j+2]) == X) &&
((sgd(vueloRealizado(this)[j]) - (sgd(vueloRealizado(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(this)[j]) - (prm(vueloRealizado(this)[j+2]) == X) &&
((sgd(vueloRealizado(this)[j]) - (sgd(vueloRealizado(this)[j+2]) == Y))

```

Estos son el invariante y la negacion de la guarda.

```

0 ≤ i ≤ |vueloRealizado(this)| - 2 && i ≥ |vueloRealizado(this)| - 2

```

```

→ i == |vueloRealizado(this)| - 2

i == |vueloRealizado(this)| - 2 && (∀ j ← [0..i)
((prm(vueloRealizado(this)[j]) - (prm(vueloRealizado(this)[j+2]) == X) &&
((sgd(vueloRealizado(this)[j]) - (sgd(vueloRealizado(this)[j+2]) == Y)))
→
(∀ 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)))

- I && Fv < cota → noB

0 ≤ i ≤ |vueloRealizado(this)| - 2 && |vueloRealizado(this)| - 2 - i < 0 &&
cumplenTodos == (∀ j ← [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 → |vueloRealizado(this)| - 2 < i
→ |vueloRealizado(this)| - 2 ≤ i

```

### 3. Listo Para Cosechar

```

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

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

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

problema _parcelasCultivoLargo (this: Sistema, (ℤ, ℤ): parcelasCultivo, ℤ: fila) = res : ℤ {
  modifica parcelasCultivo;
  asegura parcelasCultivo == [(fila, j)|j ← [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:  $0 \leq j \leq \text{largo}(\text{dimensiones}(\text{campo}(\text{this})))$  &&
// 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 == |[1| → 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  $\rightarrow$  I

i == 0  $\rightarrow$  0 == i  $\rightarrow$  0  $\leq$  i  $\leq$  0
 $\rightarrow$  0  $\leq$  i  $\leq$  |[i]|  $\rightarrow$  0  $\leq$  i  $\leq$  ancho(dimensiones(campo(this)))

listaParcelasCultivo == []  $\rightarrow$  listaParcelasCultivo = [(x,y)|x  $\leftarrow$  ([0..0]),
    y  $\leftarrow$  ([0..largo(dimensiones(campo(this))),
    contenido(campo(this), x, y) == Cultivo]
(verdadero por vacuidad del selector [0..0) )

 $\rightarrow$  listaParcelasCultivo = [(x,y)|x  $\leftarrow$  ([0..i]),
    y  $\leftarrow$  ([0..largo(dimensiones(campo(this))),
    contenido(campo(this), x, y) == Cultivo]
(Verdadero pues i == 0)

- I && noB  $\rightarrow$  Qc

i  $\leq$  ancho(dimensiones(campo(this))) && i  $\geq$  ancho(dimensiones(campo(this)))
(Invariante y NoGuarda)
 $\rightarrow$  i == ancho(dimensiones(campo(this)))

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

(Verdadero por Invariante)

 $\rightarrow$  listaParcelasCultivo = [(x,y)|x  $\leftarrow$  ([0..ancho(dimensiones(campo(this))))),
    y  $\leftarrow$  ([0..largo(dimensiones(campo(this))),
    contenido(campo(this), x, y) == Cultivo]

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

 $\rightarrow$  listaParcelasCultivo == parcelasCultivo(campo(this))  $\rightarrow$  Qc
(Reemplazo sintactico)

- (I && Fv < cota)  $\rightarrow$  noB

ancho(dimensiones(campo(this))) - i < 0
 $\rightarrow$  ancho(dimensiones(campo(this))) < i
 $\rightarrow$  i > ancho(dimensiones(campo(this)))
 $\rightarrow$  i  $\geq$  ancho(dimensiones(campo(this)))  $\rightarrow$  noB

```

---



---

Demostracion del ciclo \_parcelasCultivoLargo

```

// Pc: j == 0 && parcelasCultivoLargo == pre(parcelasCultivoLargo)
// I: 0  $\leq$  j  $\leq$  largo(dimensiones(campo(this))) &&
// parcelasCultivoLargo == pre(parcelasCultivoLargo) ++ [(fila,y)|y  $\leftarrow$  [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  $\leftarrow$  [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

```