1. Tipos

`tipo` Id = $\mathbb{Z}$;
`tipo` Carga = $\mathbb{Z}$;
`tipo` Ancho = $\mathbb{Z}$;
`tipo` Largo = $\mathbb{Z}$;
`tipo` Posicion = ($\mathbb{Z}$,$\mathbb{Z}$) ;
`tipo` Dimension = (Ancho,Largo) ;
`tipo` Parcela = Cultivo, Granero, Casa ;
`tipo` Producto = Fertilizante, Plaguicida, PlaguicidaBajoConsumo, Herbicida, HerbicidaLargoAlcance ;
`tipo` EstadoCultivo = ReciénSembrado, EnCrecimiento, ListoParaCosechar, ConMaleza, ConPlaga, NoSensado ;

2. Campo

`tipo Campo` {
`observador dimensiones` (this: Campo) : Dimension ;
`observador contenido` (p: Posicion, this: Campo) : Parcela ;
`requiere enRango` : $0 \leq prm(p) < prm(dimensiones(this)) \land 0 \leq sgd(p) < sgd(dimensiones(this))$;

`invariante dimensionesValidas` : $prm(dimensiones(this)) > 0 \land sgd(dimensiones(this)) > 0$;
`invariante unaSolaCasa` : $|[(i,j)|i \leftarrow [0..prm(dimensiones(this))), j \leftarrow [0..sgd(dimensiones(this))), contenido((i,j),this) == Casa]| == 1$;
`invariante unSoloGranero` : $|[(i,j)|i \leftarrow [0..prm(dimensiones(this))), j \leftarrow [0..sgd(dimensiones(this))), contenido((i,j),this) == Granero]| == 1$;
`invariante algoDeCultivo` : $|[(i,j)|i \leftarrow [0..prm(dimensiones(this))), j \leftarrow [0..sgd(dimensiones(this))), contenido((i,j),this) == Cultivo]| \geq 1$;
`invariante posicionesAlcanzables` : $posicionesAlcanzablesEn100(this)$;

`aux posicionesAlcanzablesEn100` (this: Campo) : Bool = $alcanzableEn100(posicionGranero(this), prm(dimensiones(this)), sgd(dimensiones(this)))$;
`aux alcanzableEn100` (posG: ($\mathbb{Z}$, $\mathbb{Z}$), a, l: $\mathbb{Z}$) : Bool = $(\forall i \leftarrow [0..a), j \leftarrow [0..l))distancia(posG,(i,j)) \leq 100$;
}

`problema Campo` (posG, posC : Posicion) = `this` : Campo {
`requiere posValidaParaGranero` : $prm(posG) \geq 0 \land sgd(posG) \geq 0$;
`requiere posValidaParaCasa` : $prm(posC) \geq 0 \land sgd(posC) \geq 0$;
`requiere casaYGraneroDistintos` : $posG \; != \; posC$;
`requiere graneroYCasaAlcanzables` : $alcanzableEn100(posG, prm(posC)+1, sgd(posC)+1)$;
`asegura hayGranero` : $enRango(dimensiones(this), prm(posG), sgd(posG)) \land contenido(posG, this) == Granero$;
`asegura hayCasa` : $enRango(dimensiones(this), prm(posC), sgd(posC)) \land contenido(posCmthis) == Casa$;

`aux distancia` (a, b: (Ancho, Largo)) : $\mathbb{Z}$ = $|prm(a) - prm(b)| + |sgd(a) - sgd(b)|$;
`aux enRango` (dim: (Ancho, Largo), i, j: $\mathbb{Z}$) : Bool = $0 \leq i < prm(dim) \land 0 \leq j < sgd(dim)$;
}

`problema dimensiones` (this: Campo) = `res` : Dimension {
`asegura` $res == dimensiones(this)$;
}

`problema contenido` (p: Posicion, this: Campo) = `res` : Parcela {
`requiere` $enRango(dimensiones(this), prm(p), sgd(p))$;
`asegura` $res == contenido(p, this)$;
}

3. Drone

```
tipo Drone {
        observador id (this: Drone) : Id ;
        observador bateria (this: Drone) : Carga ;
        observador enVuelo (this: Drone) : Bool ;
        observador vueloRealizado (this: Drone) : [Posicion] ;
        observador posicionActual (this: Drone) : Posicion ;
        observador productosDisponibles (this: Drone) : [Producto] ;

        invariante vuelosOk :
            enVuelo(this) ⇒ (|vueloRealizado(this)| > 0 ∧
            posicionActual(this) == vueloRealizado(this)_{|vueloRealizado(this)|−1} ∧
            posicionesPositivas(this) ∧ movimientosOK(this)) ∧ ¬enVuelo(this) ⇒ |vueloRealizado(this)| == 0 ;
        invariante bateriaOk : 0 ≤ bateria(this) ≤ 100 ;

        aux posicionesPositivas (d: Drone) : Bool = (∀i ← [0.. |vueloRealizado(d)|))prm(vueloRealizado(d)_i) ≥ 0 ∧
            sgd(vueloRealizado(d)_i ≥ 0 ;
        aux movimientosOK (d: Drone) : Bool = (∀i ← [1.. |vueloRealizado(d)|))
            prm(vueloRealizado(d)_i) == prm(vueloRealizado(d)_{i−1}) ∧
            (sgd(vueloRealizado(d)_i) == sgd(vueloRealizado(d)_{i−1}) − 1 ∨
            sgd(vueloRealizado(d)_i) == sgd(vueloRealizado(d)_{i−1}) + 1)
            ∨
            sgd(vueloRealizado(d)_i) == sgd(vueloRealizado(d)_{i−1}) ∧
            (prm(vueloRealizado(d)_i) == prm(vueloRealizado(d)_{i−1}) − 1 ∨
            prm(vueloRealizado(d)_{i−1}) == prm(vueloRealizado(d)_{i−1}) + 1) ;
}

problema Drone (i: ℤ, ps: [Producto]) = this : Drone {
        asegura id(this) == i ;
        asegura bateria(this) == 100 ;
        asegura ¬enVuelo(this) ;
        asegura mismos(productosDisponibles(this), ps) ;
}

problema id (this: Drone) = res : Id {
        asegura res == id(this) ;
}

problema bateria (this: Drone) = res : Carga {
        asegura res == bateria(this) ;
}

problema setBateria (this: Drone, carga : Carga) {
        requiere 0 ≤ carga ≤ 100 ;
        modifica this ;
        asegura id(this) == id(pre(this)) ;
        asegura bateria(this) == carga ;
        asegura enVuelo(this) == enVuelo(pre(this)) ;
        asegura vueloRealizado(this) == vueloRealizado(pre(this)) ;
        asegura ¬enVuelo(this) ⇒ posicionActual(this) == posicionActual(pre(this)) ;
        asegura mismos(productosDisponibles(this), productosDisponibles(pre(this))) ;
}

problema enVuelo (this: Drone) = res : Bool {
        asegura res == enVuelo(this) ;
}

problema vueloRealizado (this: Drone) = res : [Posicion] {
        asegura res == vueloRealizado(this) ;
}

problema borrarVueloRealizado (this: Drone) {
        modifica this ;
        asegura id(this) == id(pre(this)) ;
        asegura bateria(this) == bateria(pre(this)) ;
```

```
        asegura ¬enVuelo(this);
        asegura posicionActual(this) == posicionActual(pre(this));
        asegura mismos(productosDisponibles(this), productosDisponibles(pre(this)));
}

problema posicionActual (this: Drone) = res : Posicion  {
        asegura res == posicionActual(this));
}

problema cambiarPosicionActual (this: Drone, pos : Posicion)  {
        requiere ¬enVuelo(this);
        modifica this;
        asegura id(this) == id(pre(this));
        asegura bateria(this) == bateria(pre(this));
        asegura enVuelo(this) == enVuelo(pre(this));
        asegura posicionActual(this) == pos;
        asegura vueloRealizado(this) == vueloRealizado(pre(this));
        asegura mismos(productosDisponibles(this), productosDisponibles(pre(this)));
}

problema productosDisponibles (this: Drone) = res : [Producto]  {
        asegura mismos(res, productosDisponibles(this));
}

problema sacarProducto (this: Drone, p : Producto)  {
        requiere p ∈ productosDisponibles(this);
        modifica this;
        asegura id(this) == id(pre(this));
        asegura bateria(this) == bateria(pre(this));
        asegura enVuelo(this) == enVuelo(pre(this));
        asegura vueloRealizado(this) == vueloRealizado(pre(this));
        asegura ¬enVuelo(this) ⇒ posicionActual(this) == posicionActual(pre(this));
        asegura mismos(productosDisponibles(this) + +[p], productosDisponibles(pre(this)));
}

problema moverA (this: Drone, pos : Posicion)  {
        modifica this;
        asegura id(this) == id(pre(this));
        asegura bateria(this) == bateria(pre(this));
        asegura enVuelo(this);
        asegura vueloRealizado(this) == vueloRealizado(pre(this)) + +[pos];
        asegura mismos(productosDisponibles(this), productosDisponibles(pre(this)));
}

problema vueloEscalerado (this: Drone) = res : Bool  {
        asegura res == (enVuelo(this) ∧ escalerado(vueloRealizado(this)));

        aux escalerado (ps:[Posicion]) : Bool  = (∃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 vuelosCruzados (ds: [Drone]) = res : [(Posicion, $\mathbb{Z}$)] {

requiere $(\forall d \leftarrow ds)\ enVuelo(d)$;

requiere $(\forall d_1, d_2 \leftarrow ds)\ |vueloRealizado(d_1)| == |vueloRealizado(d_2)|$;

asegura sinRepetidos : $(\forall i, j \leftarrow [0..|res|),\ i \neq j)\ prm(res_i) \neq prm(res_j)$;

asegura estaOrdenada : $((\forall i, j \leftarrow [0..|res|),\ i \leq j)\ sgd(res_i) \geq sgd(res_j)) \vee$
$((\forall i, j \leftarrow [0..|res|),\ i \leq j)\ sgd(res_i) \leq sgd(res_j))$;

asegura cantidadDronesCruzados : $(\forall r \leftarrow res)\ sgd(r) == cantidadDronesCruzados(prm(r), ds)$;

asegura sonTodosCruces : $(\forall r \leftarrow res)\ sgd(r) > 1$;

asegura estanTodosLosCruces : $(\forall p \leftarrow posConCruces(ds))(\exists r \leftarrow res)\ prm(r) == p$;

aux cantidadDronesCruzados (pos: Posicion, ds: [Drone]) : $\mathbb{Z}$ =
$|[d \mid d \leftarrow ds,\ i \leftarrow [0..|vueloRealizado(d)|),\ vueloRealizado(d)_i == pos \wedge seCruzoConOtro(d, ds, i)]|$;

aux seCruzoConOtro (d: Drone, ds: [Drone], i: $\mathbb{Z}$) : Bool =
$(\exists x \leftarrow ds, \neg igualDrone(x, d))\ vueloRealizado(d)_i == vueloRealizado(x)_i$;

aux igualDrone (d, d′: Drone) : Bool $= id(d) == id(d') \wedge bateria(d) == bateria(d') \wedge enVuelo(d) == enVuelo(d')$
$\wedge\ vueloRealizado(d) == vueloRealizado(d') \wedge\ posicionActual(d) == posicionActual(d')$
$\wedge\ mismos(productosDisponibles(d), productosDisponibles(d'))$;

aux posConCruces (ds: [Drone]) : [($\mathbb{Z}$, $\mathbb{Z}$)] =
$[vueloRealizado(d)_i \mid d \leftarrow ds, i \leftarrow [0..|vueloRealizado(d)|),\ seCruzoConOtro(d, ds, i)]$;

}

4. Sistema

```
tipo Sistema {
        observador campo (this: Sistema) : Campo ;
        observador estadoDelCultivo (p Posicion, this: Sistema) : EstadoCultivo ;
            requiere enRango(dimensiones(campo(this)), prm(p), sgd(p)) ∧ contenido(p, campo(this)) == Cultivo ;
        observador enjambreDrones (this: Sistema) : [Drone] ;

        invariante identificadoresUnicos : sinRepetidos([ id(d) | d ← enjambreDrones(this) ]) ;
        invariante unoPorParcela : (∀d, d' ← dronesEnVuelo(this), id(d) ≠ id(d'))
            posicionActual(d) ≠ posicionActual(d') ;
        invariante siNoVuelanEstanEnGranero : (∀d ← enjambreDrones(this), ¬enVuelo(d))
            posicionActual(d) == posicionGranero(campo(this)) ;
        invariante siEstanEnVueloElVueloEstaEnRango : (∀d ← dronesEnVuelo(this))(∀v ← vueloRealizado(d))
            enRango(dimensiones(campo(this)), prm(v), sgd(v)) ;

        aux dronesEnVuelo (s: Sistema) : [Drone] = [ d | d ← enjambreDrones(s), enVuelo(d) ] ;
        aux sinRepetidos (xs: [T]) : Bool = (∀i, j ← [0..|xs|), i ≠ j)xs_i ≠ xs_j ;
}

problema Sistema (c: Campo, ds: [Drone]) = this : Sistema {
        requiere identificadoresUnicos : sinRepetidos([id(d)|d ← ds]) ;
        requiere bateriaMax : (∀d ∈ ds)bateria(d) == 100 ;
        requiere enTierra : (∀d ∈ ds)¬enVuelo(d) ;
        requiere enGranero : (∀d ∈ ds)posicionActual(d) == posicionGranero(c) ;
        asegura campo(this) == c ;
        asegura (∀p ← parcelasCultivo(c))estadoDelCultivo(p, this) == NoSensado ;
        asegura mismosDrones(enjambreDrones(this), ds) ;

        aux posicionGranero (c: Campo) : (ℤ, ℤ) = [ (i, j) | i ← [0..prm(dimensiones(c))), j ← [0..sgd(dimensiones(c))),
            contenido((i, j), c) == Granero ]_0 ;
        aux parcelasCultivo (c: Campo) : [(ℤ, ℤ)] =
            [(i, j) | i ← [0..prm(dimensiones(c))), j ← [0..sgd(dimensiones(c))), contenido((i, j), c) == Cultivo] ;
        aux mismosDrones (ds, es: [Drone]) : Bool = |ds| == |es| ∧ (∀d ← ds)cuentaDrone(ds, d) == cuentaDrone(es, d) ;
        aux cuentaDrone (ds: [Drone], d: Drone) : Bool = |[ x | x ← ds, igualDrone(x, d) ]| ;
}

problema campo (this: Sistema) = res : Campo {
        asegura res == campo(this) ;
}

problema estadoDelCultivo (p: Posicion, this: Sistema) = res : EstadoCultivo {
        requiere enRango(dimensiones(this), prm(p), sgd(p)) ∧ contenido(p, campo(this)) == Cultivo ;
        asegura res == estadoDelCultivo(p, s) ;
}

problema enjambreDronesS (s: Sistema) = res : [Drone] {
        asegura mismosDrones(res, enjambreDrones(s)) ;
}

problema crecer (this: Sistema) {
        modifica this ;
        asegura mismoCampo : campo(this) == campo(pre(this)) ;
        asegura mismosDrones : mismosDrones(enjambreDrones(this), enjambreDrones(pre(this))) ;
        asegura crecenSembradasYEnCrecimiento : (∀p ← parcelasCultivo(campo(this)))
            (estadoDelCultivo(p, pre(this)) == RecienSembrado
            ⇒ estadoDelCultivo(p, this)) == EnCrecimiento) ∧
            (estadoDelCultivo(p, pre(this)) == EnCrecimiento
            ⇒ estadoDelCultivo(p, this) == ListoParaCosechar) ;
        asegura elRestoQuedaIgual : (∀p ← parcelasCultivo(campo(this)))
            (estadoDelCultivo(p, pre(this)) ≠ EnCrecimiento ∧
            estadoDelCultivo(p, pre(this)) ≠ RecienSembrado)
            ⇒ estadoDelCultivo(p, this) == estadoDelCultivo(p, pre(this)) ;
}
```

problema seVinoLaMaleza (ps: [Posicion], this: Sistema) {
requiere $(\forall p \leftarrow ps)\ enRango(dimensiones(campo(this)), prm(p), sgd(p))$;
requiere $(\forall p \leftarrow ps)\ contenido(p, campo(this)) == Cultivo$;
modifica $this$;
asegura mismosDrones : $mismosDrones(enjambreDrones(this), enjambreDrones(\mathsf{pre}(this)))$;
asegura mismoCampo : $campo(this) == campo(\mathsf{pre}(this))$;
asegura seVinoLaMaleza : $(\forall p \leftarrow parcelasCultivo(campo(this)))\ p \in ps$
$\Rightarrow estadoDelCultivo(p, this) == ConMaleza$;
asegura elRestoQuedaIgual : $(\forall p \leftarrow parcelasCultivo(campo(this)))\ p \notin ps$
$\Rightarrow estadoDelCultivo(p, this) == estadoDelCultivo(p, \mathsf{pre}(this))$;
}

problema seExpandePlaga (this: Sistema) {
modifica $this$;
asegura mismosDrones : $mismosDrones(enjambreDrones(this), enjambreDrones(\mathsf{pre}(this)))$;
asegura mismoCampo : $campo(this) == campo(\mathsf{pre}(this))$;
asegura vecinosConPlaga : $(\forall p \leftarrow parcelasCultivo(campo(this)))\ vecinoConPlaga(\mathsf{pre}(this), prm(p), sgd(p))$
$\Rightarrow estadoDelCultivo(p, this) == ConPlaga$;
asegura elRestoQuedaIgual : $(\forall p \leftarrow parcelasCultivo(campo(this)))\ \neg vecinoConPlaga(\mathsf{pre}(this), prm(p), sgd(p))$
$\Rightarrow estadoDelCultivo(p, this) == estadoDelCultivo(p, \mathsf{pre}(this))$;

aux vecinoConPlaga (s: Sistema, i,j: $\mathbb{Z}$) : Bool $= enRangoConPlaga(s, i-1, j) \vee enRangoConPlaga(s, i, j+1) \vee$
$enRangoConPlaga(s, i+1, j) \vee enRangoConPlaga(s, i, j-1)$;
aux enRangoConPlaga (s: Sistema, i, j: $\mathbb{Z}$) : Bool $= enRango(dimensiones(campo(s), i, j)) \wedge$
$contenido((i, j), campo(s)) == Cultivo \wedge estadoDelCultivo((i, j), s) == ConPlaga$;
}

problema despegar (d: Drone, this: Sistema) {
requiere $(\exists x \leftarrow enjambreDrones(this))igualDrones(x, d)$;
requiere $\neg enVuelo(d)$;
requiere $bateria(d) == 100$;
requiere hayCultivoLibre : $|parcelasCultivoLibres(this)| > 0$;
modifica $this$;
asegura mismoCampo : $campo(this) == campo(\mathsf{pre}(this))$;
asegura mismoEstadoDelCultivo : $(\forall p \leftarrow parcelasCultivo(campo(this))$
$estadoDelCultivo(p, this) == estadoDelCultivo(p, \mathsf{pre}(this))$;
asegura mismaCantidadDrones : $|enjambreDrones(\mathsf{pre}(this))| == |enjambreDrones(this)|$;
asegura elDroneDespega : $(\exists x \leftarrow enjambreDrones(this))$
$id(x) == id(d) \wedge bateria(x) == bateria(d) \wedge enVuelo(x) \wedge posicionActual(x) \in parcelasCultivoLibres(this) \wedge$
$mismos(productosDisponibles(x), productosDisponibles(d))$;
asegura elRestoEsIgual : $(\forall x \leftarrow enjambreDrones(\mathsf{pre}(this)),\ \neg igualDrones(x, d))$
$(\exists y \leftarrow enjambreDrones(this))igualDrones(y, x)$;

aux parcelasCultivoLibres (s: Sistema) : [($\mathbb{Z}$, $\mathbb{Z}$)] $=$
$[p\ |p \leftarrow parcelasCultivo(campo(s)), distancia(p, posicionGranero(campo(s))) == 1 \wedge$
$(\nexists d \leftarrow dronesEnVuelo(s))\ posicionActual(d) == p]$;
}

problema listoParaCosechar (this: Sistema) = res : Bool {
asegura $res == (cantCultivosCosechables(campo(this))/|parcelasCultivo(campo(this))| \geq 0{,}9)$;

aux cantCultivosCosechables (s: Sistema) : $\mathbb{Z}$ $=$
$|[1|pos \leftarrow parcelasCultivo(campo(s)), estadoDelCultivo(pos, s) == ListoParaCosechar]|$;
}

problema aterrizarYCargarBateria (b: $\mathbb{Z}$, this: Sistema) {
requiere $0 \leq b \leq 100$;
modifica $this$;
asegura campoIgual : $campo(this) == campo(\mathsf{pre}(this))$;
asegura estadoDelcultivoIgual : $(\forall p \leftarrow parcelasCultivo(campo(\mathsf{pre}(this)))$
$estadoDelCultivo(p, this) == estadoDelCultivo(p, \mathsf{pre}(this))$;
asegura mismaCantidadDrones : $|enjambreDrones(this)| == |enjambreDrones(\mathsf{pre}(this))|$;
asegura $(\forall d \leftarrow enjambreDrones(\mathsf{pre}(this)), bateria(d) < b)cargaBateria(this, d)$;
asegura $(\forall d \leftarrow enjambreDrones(\mathsf{pre}(this)), bateria(d) \geq b)quedaIgual(this, d)$;

aux cargaBateria (s: Sistema, d: Drone) : Bool $= (\exists x \leftarrow enjambreDrones(s))id(d) == id(x) \wedge bateria(x) == 100 \wedge \neg enVuelo(x) \wedge vueloRealizado(x) == [] \wedge posicionActual(s,x) == posicionGranero(campo(s)) \wedge mismos(productosDisponibles(x), productosDisponibles(d))$;

aux quedaIgual (s: Sistema, d: Drone) : Bool $= (\exists x \leftarrow enjambreDrones(s))igualDrone(x,d)$;

}

problema fertilizarPorFilas (this: Sistema) {

requiere aLoSumoUnDroneVolandoPorFila : $(\forall f \leftarrow [0..sgd(dimensiones(campo(this))))\ |dronesVolandoEnFila(this,f)| \leq 1$;

modifica $this$;

asegura campoIgual : $campo(this) == campo(\text{pre}(this))$;

asegura mismaCantidadDrones : $|enjambreDrones(this)| == |enjambreDrones(\text{pre}(this))|$;

asegura losDelGraneroIguales : $(\forall d \leftarrow enjambreDrones(\text{pre}(this)), \neg enVuelo(d))\ quedaIgual(this,d)$;

asegura losVoladores : $(\forall d \leftarrow enjambreDrones(\text{pre}(this)),\ enVuelo(d))\ (\exists d' \leftarrow enjambreDrones(this))$
$id(d') == id(d) \wedge enVuelo(d') == enVuelo(d) \wedge$
$prm(posicionActual(d')) == prm(posicionActual(d) - recorridoMaximo(\text{pre}(this), d)) \wedge$
$sgd(posicionActual(d')) == sgd(posicionActual(d)) \wedge$
$bateria(d') == bateria(d) - recorridoMaximo(\text{pre}(this), d) \wedge$
$mismosProductosDescontandoFertilizante(\text{pre}(this), d, d')$;

asegura aLaDerechaIgual : $(\forall d \leftarrow enjambreDrones(\text{pre}(this)), enVuelo(d))$
$(\forall i \leftarrow (prm(posicionActual(d))..prm(dimensiones(campo(this)))],$
$contenido((i, sgd(posicionActual(d))), campo(this)) == Cultivo)$
$estadoDelCultivo((i, sgd(posicionActual(d))), this) == estadoDelCultivo((i, sgd(posicionActual(d))), this)$;

asegura izqNoRecorrida : $(\forall d \leftarrow enjambreDrones(\text{pre}(this)), enVuelo(d))$
$(\forall i \leftarrow [0..prm(posicionActual(d)) - recorridoMaximo(\text{pre}(this), d)),$
$contenido((i, sgd(posicionActual(d))), campo(this)) == Cultivo)$
$estadoDelCultivo((i, sgd(posicionActual(d))), this) == estadoDelCultivo((i, sgd(posicionActual(d))), \text{pre}(this))$;

asegura enRangoNoFertilizable : $(\forall d \leftarrow enjambreDrones(\text{pre}(this)), enVuelo(d))$
$(\forall i \leftarrow [prm(posicionActual(d)) - recorridoMaximo(\text{pre}(this), d)..prm(posicionActual(d))],$
$contenido((i, sgd(posicionActual(d))), campo(this)) == Cultivo)$
$estadoDelCultivo((i, sgd(posicionActual(d))), \text{pre}(this)) \notin [RecienSembrado, EnCrecimiento] \Rightarrow$
$estadoDelCultivo((i, sgd(posicionActual(d))), this) == estadoDelCultivo((i, sgd(posicionActual(d))), \text{pre}(this))$;

asegura enRangoFertilizable : $(\forall d \leftarrow enjambreDrones(\text{pre}(this)), enVuelo(d))$
$(\forall i \leftarrow [prm(posicionActual(d)) - recorridoMaximo(\text{pre}(this), d)..prm(posicionActual(d))],$
$contenido((i, sgd(posicionActual(d))), campo(this)) == Cultivo)$
$estadoDelCultivo((i, sgd(posicionActual(d))), \text{pre}(this)) \in [RecienSembrado, EnCrecimiento] \Rightarrow$
$estadoDelCultivo((i, sgd(posicionActual(d))), this) == ListoParaCosechar)$;

aux dronesVolandoEnFila (s: Sistema, f: $\mathbb{Z}$) : $\mathbb{Z}$ $=$
$[\, d \mid d \leftarrow enjambreDrones(s), enVuelo(d) \wedge prm(posicionActual(d)) == f\,]$;

aux recorridoMaximo (ps, s :Sistema, d:Drone) : $\mathbb{Z}$ $= minimo(minimo(fertAplicable(ps,s,d), bateria(d)), parcelasLibres(ps,s,d))$;

aux minimo (a, b: $\mathbb{Z}$) : $\mathbb{Z}$ $=$ if $a < b$ then a else b ;

aux fertAplicable (ps, s: Sistema, d: Drone) : $\mathbb{Z}$ $= prm(posicionActual(d)) - [i | i \leftarrow [0..prm(posicionActual(d))],$
$cantFertilizables(ps,i,d) \leq cuenta(Fertilizante, productosDisponibles(d))]_0$;

aux cantFertilizables (ps: Sistema, i:$\mathbb{Z}$, d:Drone) : $\mathbb{Z}$ $= |[1 | j \leftarrow [i..prm(posicionActual(d))),$
$estadoDelCultivo((j, sgd(posicionActual)), campo(ps)) \in [RecienSembrado, EnCrecimiento]]|$;

aux parcelasLibres (ps, s: Sistema, d: Drone) : $\mathbb{Z}$ $= prm(posicionActual(d)) - [i | i \leftarrow [0..prm(posicionActual(d))],$
$(\forall j \leftarrow [i..prm(posicionActual(d))])contenido((j, sgd(posicionActual(d)), campo(ps)) == Cultivo]_0$;

aux mismosProductosDescontandoFertilizante (ps: Sistema, pd, d: Drone) : Bool $=$
$(\forall p \leftarrow productosDisponibles(pd),\ p \neq Fertilizante)\ p \in productosDisponibles(d) \wedge$
$(\forall p \leftarrow productosDisponibles(d),\ p \neq Fertilizante)\ p \in productosDisponibles(pd) \wedge$
$cuenta(Fertilizante, productosDisponibles(d)) == cuenta(Fertilizante, productosDisponibles(pd))$
$- recorridoMaximo(ps,s,pd)$;

}

problema volarYSensarS (d: Drone, this: Sistema) {
requiere $perteneceDrone(d, enjambreDrones(this))$;
requiere $bateria(d) > 0$;
requiere hayParcelaLibre : $(\exists p \leftarrow parcelasAdyacentes(posicionActual(d), campo(this)))$
$(\nexists d' \leftarrow enjambreDrones(this))posicionActual(d') = p$;
modifica $this$;
asegura mismoCampo : $campo(this) == campo(\text{pre}(this))$;
asegura mismoCantidadDrones : $|enjambreDrones(this)| == |enjambreDones(\text{pre}(this))|$;
asegura losOtrosDronesSiguenIgual : $(\forall d' \leftarrow enjambreDrones(\text{pre}(this)), id(d) \neq id(d'))$
$perteneceDrone(d', enjambreDrones(this))$;
asegura vuelaAPosAdyacente : $(\exists d' \in ejambreDrones(this), id(d') == id(d))\ enVuelo(d') \wedge$
$posicionActual(d') \in parcelasAdyacentes(posicionActual(droneViejo(d, \text{pre}(this)))) \wedge$
$vueloRealizado(d') == vueloRealizado(droneViejo(d, \text{pre}(this))) + +[posicionActual(d')]$;
asegura seSensaOAplicaProductos :
if $contenido(p, campo(pre(this))) \neq Cultivo \vee noEstabaSensado(\text{pre}(this), posicionActual(d))$ then
$(contenido(p, campo(this)) \neq Cultivo \vee estaSensado(this, posicionActual(d))) \wedge$
$bateria(d) == bateria(droneViejo(d)) - 1 \wedge noCambioElRestoDelCultivo(\text{pre}(this), this, d)$ else
$seAplicoProducto(\text{pre}(this), this, droneViejo(d, \text{pre}(this)), d)$;

aux noEstabaSensado (s: Sistema, p: Posicion) : Bool $= \neg estaSensado(s, p)$;
aux estaSensado (s: Sistema, p: Posicion) : Bool $= estadoDelCultivo(p, s) \neq NoSensado$;
aux droneViejo (d: Drone, s: Sistema) : Drone $= [d'|d' \leftarrow enjambreDrones(s), id(d') == id(d)]_0$;
aux noCambioElRestoDelCultivo (ps, s: Sistema, d: Drone) : Bool $= (\forall p \leftarrow parcelasCultivo(campo(ps)),$
$p \neq posicionActual(d))\ igualEstadoParcela(ps, s, p)$;
aux seAplicoProducto (ps, s: Sistema, pd, d: Drone) : Bool = if $|productosUsables(ps, pd, d)| == 0$ then
$bateria(d) == bateria(pd) - 1 \wedge mismos(productosDisponibles(d), productosDisponibles(pd)) \wedge$
$cultivoIgual(ps, s)$ else
$(\exists p \leftarrow productosUsables(ps, pd, d))(bateria(d) == bateria(pd) - bateria(p) \wedge$
$mismos(productosDisponibles(d) + +[p], productosDisponibles(pd)) \wedge$
$cambioCultivo(ps, s, p, pd, d))$;
aux perteneceDrone (d:Drone, ds:[Drone]) : Bool $= ((\exists x \leftarrow ds)(igualDrone(x, d)))$;
aux parcelasAdyacentes (x:Posicion, c: Campo)) : [Posicion] $= [y|y \leftarrow parcelasCultivo(c),$
$distancia((prm(x), sgd(x)), (prm(y), sgd(y))) == 1]$;
aux productosUsables (s: Sistema, pd, d: Drone) : [Producto] $= [p|p \leftarrow productosDisponibles(pd),$
$sePuedeAplicar(p, posicionActual(d), s) \wedge bateria(pd) > bateria(p)]$;
aux sePuedeAplicar (p: Producto, pos: Posicion, s: Sistema) : Bool =
$(estadoDelCultivo(pos, s) \in [RecienSembrado, EnCrecimiento] \Rightarrow p == Fertilizante) \wedge$
$(estadoDelCultivo(pos, s) == ConMaleza \Rightarrow p \in [Herbicida, HerbicidaLargoAlcance]) \wedge$
$(estadoDelCultivo(pos, s) == ConPlaga \Rightarrow p \in [Plaguicida, PlaguicidaBajoConsumo])$;
aux bateria (p:Producto) : $\mathbb{Z}$ = if $p == Fertilizante$ then 0 else
if $p == Plaguicida$ then 10 else 5 ;
aux cultivoIgual (ps, s: Sistema) : Bool $= (\forall p \leftarrow parcelasCultivo(campo(ps)))$
$igualEstadoParcela(ps, s, p)$;
aux cambioCultivo (ps, s: Sistema, p: Producto, pd, d: Drone) : Bool =
$(\forall pos \leftarrow parcelasCultivo(campo(s)))$if $pos \in parcelasAfectadas(ps, p, d)$ then $productoAplicado(p, pos, s)$ else
$igualEstadoParcela(ps, s, pos)$;
aux parcelasAfectadas (ps: Sistema, p: Producto, d: Drone) : [Posicion] $= [pos|pos \leftarrow$
$parcelasAdyacentes(posicionActual(d), campo(ps)) + +[posicionActual(d)], sePuedeAplicar(p, pos, ps)]$;
aux productoAplicado (p: Producto, pos: Posicion, s: Sistema) : Bool = if $p == Fertilizante$ then
$estadoDelCultivo(pos, s) == ListoParaCosechar$ else
$estadoDelCultivo(pos, s) == RecienSembrado$;
aux igualEstadoParcela (ps, s: Sistema, pos: Posicion) : Bool =
$estadoDelCultivo(pos, s) == estadoDelCultivo(pos, ps)$;
}