UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación

Algoritmos y Estructura de Datos I

Primer cuatrimestre de 2016

1 de abril de 2016

TPE - Agricultura con drones

Aclaraciones

- Para aprobar la totalidad del TP es necesario tener aprobado cada uno de sus módulos.
- **No** está permitido el uso de **acum** para la resolución.

Los robots aéreos (drones) pueden utilizarse para una gran variedad de aplicaciones. En el caso de la agricultura, pueden servir para analizar el estado de los cultivos (por ejemplo, conocer en qué etapa del crecimiento se encuentran), así como también para la detección y control automático de malezas y plagas.

GreenSoft, una reconocida empresa de software que desarrolla aplicaciones para el agro, requiere de nuestros servicios para crear un nuevo sistema llamado CropFieldRobotics para la automatización de cultivos extensivos. La idea es emplear un enjambre de drones que sobrevuelen los campos y que, a partir de la información extraída por los sensores que lleva a bordo, construyan un mapa con el estado del cultivo. Este mapa contiene toda la información relevante para que el productor agropecuario haga un seguimiento de su campo hasta el momento de la cosecha. Además, los drones son capaces de aplicar herbicidas o insecticidas con gran precisión para controlar malezas y plagas que pudieran aparecer en el cultivo. De esta forma se reemplaza la fumigación con avionetas, muy común en estos días, minimizando la utilización de productos químicos y la posibilidad de contaminación ambiental.

El objetivo de este trabajo práctico es especificar problemas que componen la base del novedoso sistema CropFieldRobotics que la empresa GreenSoft pretende desarrollar.

1. Tipos Básicos

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

2. Campo

Vamos a considerar que el campo tiene una forma rectangular, cuya dimensión viene dada por su largo y por su ancho. El campo está dividido en parcelas. Una parcela puede estar destinada al cultivo, al granero o a la casa del productor.

`tipo Campo` {
`observador dimensiones` (c: Campo) : (Ancho, Largo) ;
`observador contenido` (c: Campo, i, j: $\mathbb{Z}$) : Parcela ;
`requiere enRango` : $0 \leq i < prm(dimensiones(c)) \land 0 \leq j < sgd(dimensiones(c))$;
`invariante dimensionesValidas` : $prm(dimensiones(c)) > 0 \land sgd(dimensiones(c)) > 0$;
`invariante unaSolaCasa` : $|[(i,j)|i \leftarrow [0..prm(dimensiones(c))), j \leftarrow [0..sgd(dimensiones(c))), contenido(c,i,j) == Casa]| == 1$;
`invariante unSoloGranero` : $|[(i,j)|i \leftarrow [0..prm(dimensiones(c))), j \leftarrow [0..sgd(dimensiones(c))), contenido(c,i,j) == Granero]| == 1$;
`invariante algoDeCultivo` : $|[(i,j)|i \leftarrow [0..prm(dimensiones(c))), j \leftarrow [0..sgd(dimensiones(c))), contenido(c,i,j) == Cultivo]| \geq 1$;
`invariante posicionesAlcanzables` : $posicionesAlcanzablesEn100(c)$;
}

`aux posicionesAlcanzablesEn100` (c: Campo) : Bool = $alcanzableEn100(posicionGranero(c), prm(dimensiones(c)), sgd(dimensiones(c)))$;

1. `problema crearC` (posG, posC: ($\mathbb{Z}$, $\mathbb{Z}$)) = `result` : Campo
   Devuelve un campo que contiene el granero en la posición *posG* y la casa en la posición *posC*.
2. `problema dimensionesC` (c: Campo) = `result` : (Ancho, Largo)
   Devuelve las dimensiones del campo *c*.

3. `problema contenidoC` (c: Campo, i, j: $\mathbb{Z}$) = `result` : Parcela
   Devuelve el contenido de la parcela en la posición (i, j).

3. Drone

Para simplificar el modelo vamos a asumir que los drones realizan trayectorias sobre el campo siempre de manera recta, avanzando en cada momento de una parcela a su adyacente. Consideramos como parcela adyacente a otra a aquella parcela que está en alguno de los cuatro puntos cardinales en línea recta respecto de la primera. Es decir, dada una parcela que se encuentra en posición (i, j), las parcelas adyacentes son las que se encuentran en las posiciones $(i-1, j), (i+1, j), (i, j-1)$ y $(i, j+1)$, siempre y cuando estas posiciones estén dentro de las dimensiones del campo donde opera el drone. Cada movimiento del drone decrementa en 1 % la carga de batería de la que dispone. Cuando el drone se queda sin batería debe aterrizar para volver a cargarla, como se detallará más adelante. Los drones son capaces de sensar la parcela que sobrevuelan, y también aplicar distintos productos sobre el cultivo. El observador *productosDisponibles* indica el *stock* utilizable que tiene cada drone. Por ejemplo, un drone que cuya lista de productos disponibles es [*Plaguicida*, *Herbicida*, *Plaguicida*] podrá aplicar plaguicida sobre el cultivo de dos parcelas, y herbicida sobre una. Los drones que especificaremos en este trabajo práctico tienen una autonomía de vuelo para recorrer hasta 100 parcelas. La autonomía puede disminuir si aplican productos químicos, como se describirá más adelante. Cada drone tiene un identificador unívoco. Mientras los drones se encuentran en vuelo, se puede conocer la trayectoria realizada (el historial de posiciones recorridas) durante ese vuelo mediante el observador *vueloRealizado*.

`tipo Drone` {
  `observador id` (d: Drone) : Id ;
  `observador bateria` (d: Drone) : Carga ;
  `observador enVuelo` (d: Drone) : Bool ;
  `observador vueloRealizado` (d: Drone) : [($\mathbb{Z}$, $\mathbb{Z}$)] ;
  `observador posicionActual` (d: Drone) : ($\mathbb{Z}$, $\mathbb{Z}$) ;
  `observador productosDisponibles` (d: Drone) : [Producto] ;

  `invariante vuelosOk` :
    $enVuelo(d) \Rightarrow (|vueloRealizado(d)| > 0 \land posicionActual(d) == vueloRealizado(d)_{|vueloRealizado(d)|-1} \land$
    $posicionesPositivas(d) \land movimientosOK(d)) \land \neg enVuelo(d) \Rightarrow |vueloRealizado(d)| == 0$;
  `invariante bateriaOk` : $0 \leq bateria(d) \leq 100$;
}

`aux posicionesPositivas` (d: Drone) : Bool $= (\forall i \leftarrow [0..|vueloRealizado(d)|))prm(vueloRealizado(d)_i) \geq 0 \land$
$sgd(vueloRealizado(d)_i \geq 0$;
`aux movimientosOK` (d: Bool) : Bool $= (\forall i \leftarrow [1..|vueloRealizado(d)|))$
$prm(vueloRealizado(d)_i) == prm(vueloRealizado(d)_{i-1}) \land (sgd(vueloRealizado(d)_i) == sgd(vueloRealizado(d)_{i-1}) - 1 \lor$
$sgd(vueloRealizado(d)_i) == sgd(vueloRealizado(d)_{i-1}) + 1) \lor sgd(vueloRealizado(d)_i) == sgd(vueloRealizado(d)_{i-1})$
$\land (prm(vueloRealizado(d)_i) == prm(vueloRealizado(d)_{i-1}) - 1 \lor prm(vueloRealizado(d)_{i-1}) == prm(vueloRealizado(d)_{i-1}) +$
1) ;

4. `problema crearD` (id: $\mathbb{Z}$, pd: [Producto]) = `result` : Drone
   Crea un drone que no está en vuelo con id *id*, productos disponibles *pd* y batería máxima.
5. `problema idD` (d: Drone) = `result` : $\mathbb{Z}$
   Devuelve el identificador del drone *d*.
6. `problema bateriaD` (d: Drone) = `result` : $\mathbb{Z}$
   Devuelve el nivel de batería del drone *d*.
7. `problema enVueloD` (d: Drone) = `result` : Bool
   Devuelve un booleano indicando si el drone *d* se encuentra volando.
8. `problema vueloRealizadoD` (d: Drone) = `result` : [($\mathbb{Z}$, $\mathbb{Z}$)]
   Si el drone *d* se encuentra volando, devuelve la lista de las posiciones de las parcelas que recorrió estando en vuelo (no se considera la parcela desde la cual despega).
9. `problema posicionActualD` (d: Drone) = `result` : ($\mathbb{Z}$, $\mathbb{Z}$)
   Devuelve la posición de la parcela actual donse se encuentra el drone *d*.
10. `problema productosDisponiblesD` (d: Drone) = `result` : [Producto]]
    Devuelve los productos que puede utilizar el drone *d*.
11. `problema vueloEscaleradoD` (d: Drone) = `result` : Bool
    Devuelve verdadero sii el drone *d* se encuentra volando y el vuelo que está realizado tiene forma de *escalera*, es decir que sus movimientos alternan filas y columnas, siempre incrementándose o siempre decrementándose de a uno.

12. `problema vuelosCruzadosD` (ds: [Drone]) = `result` : [((ℤ, ℤ), ℤ)]
Dada una lista de drones en vuelo, todos con vuelos igual de largos, devuelve otra lista con aquellas posiciones en las que se cruzaron varios drones. Se considera que $n > 0$ drones se cruzan si y sólo si en el mismo momento (mismo índice en la lista de vuelos) pasan por la misma posición del campo. En la lista devuelta, la primer coordenada debe contener las posiciones en las que hubo cruces de drones. En la segunda coordenada se debe indicar cuántos drones formaron parte del cruce. La lista resultante debe estar ordenada de acuerdo a la segunda coordenada (cantidad de drones que se cruzaron en cierta posición). Cabe aclarar que por lo general se intentan no tener los vuelos cruzados para evitar colisiones entre drones.

4. Sistema

El objetivo del sistema es explorar el campo utlizando un conjunto o enjambre de drones, aplicar herbicida en caso de que haya malezas e insecticidas en caso de que haya plagas, y poder distinguir cuándo el cultivo del campo está listo para ser cosechado. Si los drones se quedan sin carga de batería deben aterrizar en el granero del campo para recargarla. Llegado el caso de perder toda la carga de la batería sin haber podido alcanzar el granero los drones pueden planear, lo que les permite únicamente desplazarse (sin poder aplicar productos ni sensar). Una vez que recargan totalmente su batería pueden despegar, comenzando su vuelo sobre cualquier parcela adyacente al granero (recordar definición de parcela adyacente) al granero en la cual no haya otro drone, ya que en este sistema nunca puede haber más de un drone sobrevolando la misma parcela (vuelos cruzados) para evitar colisiones. Los drones pueden utilizar los productos que tienen disponibles cuando se encuentran sobre parcelas de tipo *Cultivo*, y el efecto de cada producto es el siguiente:

- *Fertilizante*: Si el cultivo está en estado *RecienSembrado* o *EnCrecimiento*, lo deja en estado *ListoParaCosechar*. Utilizar fertilizante no consume batería del drone.
- *Plaguicida*: En caso que el cultivo esté en estado *ConPlaga*, el plaguicida lo transforma en estado *RecienSembrado*. Se consume 10 % de la carga de la batería del drone para liberar este producto.
- *PlaguicidaBajoConsumo*: Tiene el mismo efecto que el Plaguicida pero sólo consume el 5 % de la carga de la batería del drone que lo utiliza.
- *Herbicida*: En caso de que el cultivo esté en estado *ConMaleza*, el herbicida lo transforma en estado *RecienSembrado*. Se consume el 5 % de la carga de la batería del drone para liberar este producto.
- *HerbicidaLargoAlcance*: En caso que el cultivo o alguna de sus parcelas adyacentes que también sean *Cultivo* esté en estado *ConMaleza*, el herbicida lo transforma en estado *RecienSembrado*. Se consume 5 % de carga de la batería del drone para liberar este producto.

```
tipo Sistema {
      observador campo (s: Sistema) : Campo ;
      observador estadoDelCultivo (s: Sistema, i, j: ℤ) : EstadoCultivo ;
           requiere enRango(dimensiones(s), i, j) ∧ contenido(campo(s), i, j) == Cultivo ;
      observador enjambreDrones (s: Sistema) : [Drone] ;

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

13. `problema crearS` (c: Campo, ds: [Drone]) = `result` : Sistema
Crea un sistema con los drones de la lista *ds* y el campo *c*. El sistema debe contener a los drones en el granero con la batería cargada al máximo, y todas las parcelas que tienen cultivo debe encontrarse en estado *NoSensado*.

14. `problema campoS` (s: Sistema) = `result` : Campo
Devuelve el campo sobre el cual funcionará el sistema *s*.

15. `problema estadoDelCultivoS` (s: Sistema, i, j: ℤ) = `result` : EstadoCultivo
Devuelve el estado del cultivo en la posición (i, j).

16. `problema enjambreDronesS` (s: Sistema) = `result` : [Drone]
Devuelve los drones del sistema *s*.

17. `problema crecerS` (s: Sistema)
Modifica el campo *c* de forma que cada *Parcela* de *Cultivo* evolucione de la siguiente manera:

- Si está *RecienSembrado* pasa a estar *EnCrecimiento*.
- Si está *EnCrecimiento* pasa a estar *ListoParaCosechar*.
- Si está en algún otro estado no se modifica.

18. `problema seVinoLaMalezaS` (s: Sistema, ps: [($\mathbb{Z}$, $\mathbb{Z}$)])
Modifica el sistema de forma tal que todas las parcelas de cultivo de la lista *ps* pasen a estar $ConMaleza$.

19. `problema seExpandePlagaS` (s: Sistema)
Modifica el sistema de forma tal que todas las parcelas de cultivo adyacentes a las parcelas con plaga pasan también a tener plaga.

20. `problema despegarS` (s: Sistema, d: Drone)
Modifica el sistema d de forma tal que el drone d que se encontraba recargando batería en el granero comience un nuevo vuelo.

21. `problema listoParaCosecharS` (s: Sistema) = `result` : Bool
Devuelve True sii al menos el 90 % de las *Parecelas* de *Cultivo* están en estado *ListoParaCosechar*.

22. `problema aterrizarYCargarBateriaS` (s: Sistema, b: $\mathbb{Z}$)
Modifica el sistema haciendo que todos los drones que tengan el nivel de batería por debajo de b se posicionen en el granero. Al hacer esto, sus baterías se cargan al 100 % y sus historiales de vuelo se limpian.

23. `problema fertilizarPorFilas` (s: Sistema)
Dado un sistema en el cual no hay más de un drone en vuelo en cada fila del campo, los hace avanzar a todos hacia el *oeste* aplicando todo el fertlizante que posean. Cada drone finaliza su recorrido cuando se le agota la batería, se queda sin fertilizante, llega a una parcela que no de cultivo o se termina el campo.

24. `problema volarYSensarS` (s: Sistema, d: Drone)
Modifica el sistema s haciendo que el drone d se mueva a una parcela adyacente de donde estaba. Si el drone se mueve a una *Parcela* de *Cultivo* cuyo estado es *NoSensado* sensa esa parcela. En caso contrario, de acuerdo a los productos de los que disponga realiza alguna de las siguientes acciones:

- Si tiene algún tipo de herbicida, y la parcela a la cual el drone llega es de *Cultivo* y está $ConMaleza$, aplica el herbicida y planta semillas, dejando el cultivo en estado *RecienSembrado*.
- Si tiene algún tipo de plaguicida, y la parcela a la cual el drone llega es de *Cultivo* y está $ConPlaga$, el drone aplica el plaguicida y planta semillas, dejando el cultivo en estado *RecienSembrado*.

En todos los casos, es necesario que el drone cuente con suficiente batería como para utilizar los productos. En cualquier caso, luego de utilizar un producto, éste se elimina del listado de productos disponibles en el drone.

5. Auxiliares

`aux alcanzableEn100` (posG: ($\mathbb{Z}$, $\mathbb{Z}$), a, l: $\mathbb{Z}$) : Bool $= (\forall i \leftarrow [0..a), j \leftarrow [0..l))distancia(posG, (i, j)) \leq 100$;
`aux posicionGranero` (c: Campo) : ($\mathbb{Z}$, $\mathbb{Z}$) $= [\,(i, j)\,|\,i \leftarrow [0..prm(dimensiones(c))), j \leftarrow [0..sgd(dimensiones(c))), contenido(c, i, j) == Granero\,]_0$;
`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 $= 0 \leq i < prm(dim) \wedge 0 \leq j < sgd(dim)$;
`aux sinRepetidos` (xs: [T]) : Bool $= (\forall x \leftarrow xs)cuenta(x, xs) == 1$;