Proyecto de Simulación y Programación Declarativa: Agentes

Curso 2020-2021

- Darian Dominguez Alayón C-411

Introducción

En este proyecto se ponen de manifiesto los contenidos aprendidos sobre los agentes, los cuales fueron vistos en la asignatura de Simulación. Un agente es un sistema computacional situado dentro de un medio ambiente, dentro del cual es capaz de realizar acciones autónomas encaminadas a lograr sus objetivos. Apoyándonos en este concepto, en todo el contenido que trae consigo y utilizando el lenguaje de programación Haskell, estudiado en la asignatura Programación Declarativa, se propone darle solución al siguiente problema.

Orden del problema asignado

Marco general

El ambiente en el cual intervienen los agentes es discreto y tiene la forma de un rectángulo de N x M. El ambiente es de información completa, por tanto, todos los agentes conocen toda la información sobre el agente. El ambiente puede variar aleatoriamente cada t unidades de tiempo. El valor de t es conocido.

Las acciones que realizan los agentes ocurren por turnos. En un turno, los agentes realizan sus acciones, una sola por cada agente, y modifican el medio sin que este varíe a no ser que cambie por una acción de los agentes. En el siguiente, el ambiente puede variar. Si es el momento de cambio del ambiente, ocurre primero el cambio natural del ambiente y luego la variación aleatoria. En una unidad de tiempo ocurren el turno del agente y el turno de cambio del ambiente.

Los elementos que pueden existir en el ambiente son obstáculos, suciedad, niños, el corral y los agentes que son llamados Robots de Casa. A continuación se precisan las características de los elementos del ambiente:

Obstáculos: Estos ocupan una única casilla en el ambiente. Ellos pueden ser movidos, empujándolos, por los niños, una única casilla. El Robot de Casa, sin embargo, no puede moverlo. No pueden ser movidos por ninguna de las casillas ocupadas por cualquier otro elemento del ambiente.

Suciedad: La suciedad es por cada casilla del ambiente. Solo puede aparecer en casillas que previamente estuvieron vacías. Esta, o aparece en el estado inicial o es creada por los niños.

Corral: El corral ocupa casillas adyacentes en número igual al del total de niños presentes en el ambiente. El corral no puede moverse. En una casilla del corral solo puede coexistir un niño. En una casilla del corral, que esté vacía, puede entrar un robot. En una misma casilla del corral pueden coexistir un niño y un robot solo si el robot lo carga, o si acaba de dejar al niño.

Niño: Los niños ocupan solo una casilla. Ellos en el turno del ambiente se mueven, si es posible (si la casilla no está ocupada: no tiene suciedad, no está el corral, no hay un Robot de Casa), y aleatoriamente (puede que no ocurra movimiento), a una de las casillas adyacentes. Si esa casilla está ocupada por un obstáculo este es empujado por el niño, si en la dirección hay más de un obstáculo, entonces se desplazan todos. Si el obstáculo está en una posición donde no puede ser empujado y el niño lo intenta, entonces el obstáculo no se mueve y el niño ocupa la misma posición. Los niños son los responsables de que aparezca suciedad. Si en una cuadrícula de 3 por 3 hay un solo niño, entonces, luego de que él se mueva aleatoriamente, una de las casillas de la cuadrícula anterior que esté vacía puede haber sido ensuciada. Si hay dos niños se pueden ensuciar hasta 3. Si hay tres niños o más pueden resultar sucias hasta 6. Los niños cuando están en una casilla del corral, ni se mueven ni ensucian. Si un ni no es capturado por un Robot de Casa tampoco se mueve ni ensucia.

Robot de Casa: El Robot de Casa se encarga de limpiar y de controlar a los niños. El Robot se mueve a una de las casillas adyacentes, las que decida. Solo se mueve una casilla si no carga un niño. Si carga un niño puede moverse hasta dos casillas consecutivas. También puede realizar las acciones de limpiar y cargar niños. Si se mueve a una casilla con suciedad, en el próximo turno puede decidir limpiar o moverse. Si se mueve a una casilla donde está un niño, inmediatamente lo carga. En ese momento, coexisten en la casilla Robot y niño. Si se mueve a una casilla del corral que está vacía, y carga un niño, puede decidir si lo deja esta casilla o se sigue moviendo. El Robot puede dejar al niño que carga en cualquier casilla. En ese momento cesa el movimiento del Robot en el turno, y coexisten hasta el próximo turno, en la misma casilla, Robot y niño.

El objetivo del Robot de Casa es mantener la casa limpia. Se considera la casa limpia si el 60 % de las casillas vacías no están sucias.

Principales Ideas seguidas para la solución del problema

- En la orden del problema se presenta un ambiente discreto lo que significa que existe un número fijo y finito de percepciones y acciones que lo pueden modificar.

- En el ambiente, el agente tiene toda la información sobre el estado actual y puede tormar decisiones en correspondencia a la percepción que crea que es mejor seguir. Debido a la aleatoriedad del ambiente cada cierto tiempo t, el agente no podrá tomar decisiones pensando en el futuro.
- El ambiente es dinámico, pues este varía cada cierto intervalo de tiempo t y dichos cambios no están en control del agente. Los principales responsables de esto son los niños los cuales pueden moverse hacia una casilla adyacente, dejar suciedad en alguna casilla o empujar obstáculos. Los responsables de limpiar y controlar a los niños son los robots (agentes).
- Los agentes pueden realizar las acciones de cargar niños, limpiar suciedad, moverse a casillas adyacentes sin niño cargado, moverse a casillas adyacentes uno o dos pasos con niño cargado y dejar un niño en cualquier casilla posible a excepción de una con suciedad.
- Los agentes pueden moverse a una casilla vacía, con corral vacío o con suciedad. También pueden moverse a casillas que contengan niños, si el agente no está cargando uno. No se permite que un agente pueda limpiar con un niño en la mano.
- Como percepciones del ambiente del agente podemos destacar el camino al niño más cercano, al corral más cercano, a la casilla sucia más cercana y el por ciento de limpieza del ambiente.
- Tenemos dos estados para nuestros agentes. Estos pueden estar sin cargar niños o cargando niños.
- Nuestro agente tiene varias funcionalidades como son limpiar las casillas sucias, llevar a los niños al corral o en caso de no ser posible ninguna de estas entonces moverse aleatoriamente. El robot antes de realizar una acción analiza el ambiente:
  - Si el por ciento de casillas vacías que no están sucias es menor que 75 entonces el robot limpiará las casillas que estén sucias hasta que el por ciento dicho anteriormente vuelva a estar por encima de 75.
  - Si el por ciento de casillas vacías que no están sucias es mayor que 75 entonces el robot tratará de llevar al niño más cercano al corral más cercano.
  - Si en un momento cualquiera el robot no tiene que buscar niños ni limpiar las casillas sucias entonces se moverá con un comportamiento random para alguna de las casillas adyacentes a él.

Modelos de Agentes considerados (por cada agente se deben presentar dos modelos diferentes)

Como ya se mencionó anteriormente el ambiente del problema es dinámico por lo que cada cierto tiempo t el ambiente va a cambiar. Los principales responsables de esto son los niños que pueden moverse a una casilla adyacente, empujar obstáculos y ensuciar casillas. También los robots al ejecutar sus acciones cambian el ambiente. Debido a este dinamismo no podemos considerar que

nuestro agente sea puramente pro-activo, pues si el ambiente cambia y las precondiciones se vuelven falsas durante el proceso, entonces el comportamiento del procedimiento se indefine o simplemente falla. En la orientación del problema también se plantea que el agente conoce toda la información del ambiente, por lo que puede tomar las decisiones en el momento actual del ambiente. Gracias a esto el agente es capaz de percibir los cambios en el ambiente y reaccionar de modo oportuno a los mismos con el fin de alcanzar sus objetivos, por lo que podemos decir que el agente es reactivo. No queremos un agente puramente reactivo para que el robot no esté reaccionando constantemente y pueda enfocarse en un objetivo un determinado tiempo hasta lograr cumplirlo. Por tanto nuestro robot se basa en una unión entre agente pro-activo y reactivo.

Ideas seguidas para la implementación

Para la implementación nos basamos en todo el razonamiento anteriormente explicado. A continuación explicaremos el código el cual aparece en la carpeta code:

Main.hs: Este es el archivo principal de la aplicación. Lo primero que hace es recibir los valores requeridos por el problema como son: número de casillas con obstáculos, número de casillas sucias, número de casillas con niños, número de robots, tiempo que demora en cambiar el ambiente, número de turnos que va a demorar la simulación y cantidad de simulaciones. A continuación lo que se hace es crear las listas de dichos elementos con posiciones aleatorias en el ambiente. Estas listas son usadas en la función de simulación, la cual va a ser la encargada de controlar todo el proceso. Esta función se ejecutará hasta que se alcancen los turnos que se pasaron en la entrada y una vez hecho esto se devolverá si el ambiente quedó limpio o sucio (el límite es 60 %), el por ciento de limpieza alcanzado y si quedó algún niño libre (que no está ni en el corral ni lo carga un robot) o no. Recordar que en un turno ocurre una acción de cada agente. Si el módulo de la cantidad de turnos por la que va corriendo el programa con respecto al tiempo que demora en cambiar el ambiente es igual a 0, entonces este cambia. De lo contrario el agente i-ésimo ejecutará la acción que crea que es la correcta. Una vez terminado lo anterior para cada simulación se devolverá un por ciento promedio de limpieza del ambiente y si este por ciento representa que quedó limpio o sucio. Destacar que cada vez que se ejecute la aplicación los resultados se guardarán en *output.txt* por si se desean consultar posteriormente

En la carpeta *Modules* aparecen los siguientes módulos en los cuales se apoya la aplicación:

Elements.hs: Este módulo contiene al tipo Element el cual tiene un nombre *name* y una posición *position* para ubicar los elementos. Este módulo contiene todas las funciones relacionadas con los elementos como son: *create_element*, *move_element*, etc.

Environment.hs: Módulo que se encarga de gestionar todo lo que tiene que ver con el ambiente como crear las listas de los elementos, mover los elementos, pintar y mostrar el ambiente en consola y otras funcionalidades que son importantes en el ambiente.

Robots.hs: Como su propio nombre indica es para hacer todo lo relacionado

con los robots (agentes). Aquí podemos ver todas las acciones que realiza el robot: cargar niño, llevar niño al corral, limpiar las casillas sucias, etc. También está implentado el *robot_random* que se ejecutará cuando el robot no tenga opciones de limpiar ni de cargar niños.

Utils.hs: Aquí tenemos funciones que usamos en varios lugares del código permitiendo así la refactorización. Por ejemplo aquí tenemos las funciones que tienen que ver con el random, las cuales son primordiales en el proyecto.

Consideraciones obtenidas a partir de la ejecución de las simulaciones del problema

Se probó varias veces la aplicación con diferentes parámetros y en la mayoría de los casos se mostró un resultado positivo en cuanto a la limpieza del ambiente gracias al trabajo de los robots. A continuación presentamos dos ejemplos :

```
<n> <m> <# obstacles> <# dirty> <# childrens> <# robots> <t_change_env> <# turns> <# simulations>
5 5 2 4 3 2 2 10 5

Sim: 1
Some child are free
Environment clean
Clean percent: 78%

Sim: 2
Some child are free
Environment clean
Clean percent: 83%

Sim: 3
Some child are free
Environment clean
Clean percent: 83%

Sim: 4
Some child are free
Environment clean
Clean percent: 84%

Sim: 5
Some child are free
Environment clean
Clean percent: 80%
----------------------------------------------------------------------------------------------------

FINAL RESULTS
Environment clean
Clean Percent Average: 81%
```

En este ejemplo podemos ver como en las simulaciones se obtuvo un buen por ciento de limpieza lo que provocó que el promedio de limpieza fuese positivo también. Cabe destacar que en cada simulación se quedó al menos algún niño libre una vez pasado el la cantidad de turnos, que en este caso fue 10.

```
<n> <m> <# obstacles> <# dirty> <# childrens> <# robots> <t_change_env> <# turns> <# simulations>
4 4 1 4 4 2 1 7 5

Sim: 1
Some child are free
Environment dirty
Clean percent: 42%

Sim: 2
Some child are free
Environment dirty
Clean percent: 57%

Sim: 3
Some child are free
Environment clean
Clean percent: 83%

Sim: 4
Some child are free
Environment clean
Clean percent: 85%

Sim: 5
Some child are free
Environment dirty
Clean percent: 33%
-----------------------------------------------------------------------------------------------------

FINAL RESULTS
Environment clean
Clean Percent Average: 60%
```

Por otro lado podemos ver en este ejemplo que en la mayoría de las simulaciones fue por debajo del 60 % por lo que el abiente en esos casos estuvo sucio. Sin embargo se observa que el promedio de limpieza fue exactamente del 60 % lo que implica que estuvo limpio, rozando el borde, el promedio de limpieza de las simulaciones. Aquí también se quedó al menos algún niño libre una vez pasados los 7 turnos.

Ejecución

Para ejecutar la aplicación hay que tener instalado algún compilador de haskell como *ghci*, ejecutarlo en la carpeta *code*, luego escribir en la consola *:l Main.hs* y más tarde escribir *main*. Otra forma de correr la aplicación es instalar *stack*, luego usando *stack* instalar *runhaskell* y por último ejecutar *runhaskell Main.hs* dentro de la carpeta *code*. Esto funciona perfecto en Linux. En Windows puede dar problemas con el módulo *System.Random* por lo que se recomienda el uso de *stack ghci*. Una vez ejecutada la aplicación solo quedará ingresar el número de casillas con obstáculos, número de casillas sucias, número de casillas con niños, número de robots, tiempo que demora en cambiar el ambiente, número de turnos que va a demorar la simulación y cantidad de simulaciones .

Link del proyecto en GitHub

https://github.com/Darian10/Proyecto-de-Simulacion-y-Haskell.git