Tile World
Proyecto de simulación escrito en haskell

Marcos Adrián Valdivié Rodríguez C-412

February 4, 2022

1. Orden del Proyecto

Cada estudiante debe realizar la implementación del problema presentado. Esta implementacioón se debe realizar en Haskell y la solución debe estar en un repositorio de Github. La URL del proyecto debe entregarse por correo electrónico al profesor de conferencia de Simulación (yudy@matcom.uh.cu) con copia a la profesora de conferencia de Programación Declarativa (dafne@matcom.uh.cu), antes del viernes 28 de enero a las 12:00 de la noche. Junto a la implementación, en el proyecto de Github, debe estar presente el informe del trabajo (un documento en formato pdf). El informe de trabajo debe contener los siguientes elementos:

- Generales del Estudiante
- Orden del Problema Asignado
- Principales Ideas seguidas para la solución del problema
- Modelos de Agentes considerados (por cada agente se deben presentar dos modelos diferentes)
- Ideas seguidas para la implementación
- Consideraciones obtenidas a partir de la ejecución de las simulaciones del problema (determinar a partir de la experimentación cuáles fueron los modelos de agentes que mejor funcionaron)

1.1 Descripción del problema

El ambiente en el cual intervienen los agentes es discreto y tiene la forma de un rectángulo de NxM. 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 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 casilla 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 cuadrıícula 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 casillas. 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 niño 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 sino carga un niño. Si carga un niño pude 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 estaá 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 vacıía, 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.

2. Principales ideas para la implementación del problema

Se explicará las principales idea seguidas para la implementación de cada uno de los módulos que conforman la solución.

Módulo *Configuration.hs*

Este módulo guarda los valores de configuración que son usados para la construcción del tablero del ambiente, se decidió implementarlo en un módulo separado y con la mínima cantidad de lógica necesaria para que pueda ser usado para cambiar los parámetros desde cualquier editor de textos. En este módulo se guardan los valores correspondientes a la cantidad de niños, robots, el intervalo de cambio aleatorio del ambiente y las magnitudes del tablero y del corral.

Módulo *Random.hs*

Este módulo contiene la lógica para la simulación de las variables aleatorias utilizadas durante la ejecución del problema.

- El Monad rand recibe un entero m como parámetro y se usa para simular una variable aleatoria discreta entre 0 y m-1.
- El Monad ber se utiliza para simular una variable aleatoria discreta (0,1).
- El Monad randomSelect se utiliza para escoger, dado un Int n y una lista, n elementos aleatorios distintos de dicha lista.

Toda la simulación de variables aleatorias se basa en la obtención de la cantidad de milisegundos del reloj de la máquina utilizando la función 'Data.Time.Clock(getCurrentTime)'.

Módulo *Init.hs*

Este módulo se encarga de la inicialización del tablero del ambiente.

- El Monad initBoard se encarga de crear una nueva instancia del tablero con la cantidad de elementos de cada tipo especificado en el módulo 'Configuration.hs'. Es necesario destacar que las casillas de la esquina superior izquierda del tablero estarán siempre dedicadas al Corral, y los demás elementos serán distribuidos aleatoriamente en las casillas restantes.
- El Monad shuffle se encarga de modificar de manera aleatoria los elementos del tablero, esto es usado para simular la variación aleatoria del tablero cada cierto tiempo.
- La función validateBoard se utiliza para verificar que los parámetros dados para la construcción del tablero inicial satisfacen las condiciones del problema, es decir, que la cantidad de casillas del corral sea mayor igual que la cantidad de niños y que el total de elementos iniciales sea menor igual que la cantidad de casillas del tablero.

Módulo *Types.hs*

Este módulo contiene las definiciones de los tipos que son usados en el programa y define la forma de imprimirlos al implementar la clase Eq de cada uno.

- El tipo Matrix a que es un alias del tipo Array (Int,Int) a definido en el módulo Data.Array. Además se definieron funciones para extraer información de dichas matrices como su tamaño.
- El tipo Cell que se utiliza para representar el estado de cada una de las casillas del tablero.
- El ambiente es representado en el programa con el tipo Board como un alias a 'Matrix Cell'.

Se definieron dos tipos de robots, StateRobot y ReactiveRobot, siguiendo dos modelos de agentes distintos.

Para la representación en consola de cada uno de los elementos del tablero se siguió el siguiente patrón:

- Obstacle = #
- Muck = $*$
- Child = C
- ChildHouse = B
- ReactiveRobot = R
- ChildReactiveRobot = %
- MuckReactiveRobot = r
- StateRobot = S
- ChildStateRobot = %
- MuckStateRobot = s

Módulo *Moves.hs*

Este módulo consta de tres funciones básicas para la modificación del tablero.

- moveTo b p e mueve el elemento que está en la posición p del tablero b a la posición e. Si en la posición e hay algún elemento distinto de Empty, entonces el uso de este método ocasionaría que se perdiera la información sobre dicho elemento.
- put b p x coloca en la posición p del tablero un elemento de tipo x, sobrescribiendo el elemento que había anteriormente.
- clean b p coloca en la posición p del tablero b el elemento Empty, sobrescribiendo el elemento que había anteriormente.

Módulo *Utils.hs*

Este módulo posee la lógica básica para el movimiento dentro del tablero.

- La función insideBoard que retorna si la posición actual se encuentra o no dentro del tablero.
- El tipo Pos que es un alias a (Int,Int) y se utiliza para representar las distintas posiciones o casillas del tablero.
- El tipo Mov que puede ser L, R, U, D, representando izquierda, derecha, arriba y debajo y define las distintas direcciones en que se puede mover un elemento del tablero.
- Las funciones adjacents and around que se utilizan para obtener dada una posición las casillas a los que se puede mover y las casillas que se encuentran alrededor de la posición dada respectivamente.
- La función oo que se utiliza para obtener un Int con valor bien grande que pueda ser usado como comodín representando al infinito.

Módulo *Main.hs*

Este módulo posee el ciclo principal del programa, el cual se encarga de:

- Crear el tablero.
- Llamar a las funciones necesarias para cambiar el ambiente y mover a los niños y los robots.
- Chequea si la aplicación debe detenerse, cuando ya todos los niños y suciedad fueron controlados.

- Imprimir en consola la información necesaria para controlar el funcionamiento del programa. Cada vez que se mueven los robots y los niños se imprime el nuevo tablero. Además, se calcula tras cada paso el porcentaje de suciedad y se imprime este, así como el máximo valor que tuvo durante toda la ejecución del programa al finalizar el mismo.

Módulo *Distances.hs*

Este módulo posee los algoritmos básicos para poder tomar decisiones sobre que camino tomar por cada robot.

- La función distanceMatrix recibe un tablero del ambiente, una posición y un conjunto de posibles obstáculos y calcula la distancia desde esa posición hasta el resto de las casillas del tablero respetando dichos obstáculos. Esta función efecúa un BFS en las posiciones del tablero alcanzables desde la posición inicial y retorna un objeto Matrix Int con los cálculos hechos.
- La función buildPath recibe dos posiciones del tablero y retorna el camino más corto entre estas, haciendo uso del array de distancias que se le pasa como parámetro.
- La función nearest calcula el objeto más cercano de un tipo dado haciendo uso del array de distancias que se le pasa como parámetro.
- La función pathExists calcula la distancia entre una posición del tablero y el Corral.

Módulo *ChildMovements.hs*

Este módulo es el encargado de manejar el movimiento de los niños. Estos se mueven aleatoriamente a alguna de las cuatro casillas adyacentes, siempre que sea posible, y dejando rastros de suciedad, también de forma aleatoria, según la cantidad de niños que hayan cerca. El Monad moveChild, que se encarga de mover un niño en particular, se apoya en el módulo 'Random.hs' para la simulación de la variable aleatoria cada vez que sea necesario.

Módulo *RobotMovements.hs*

Este módulo es el encargado de identificar que método en específico es el encargado de mover a cada robot, y llamarlo en consecuencia.

Módulo *ReactiveRobot.hs*

Este módulo posee la lógica para el funcionamiento de los robots reactivos. Estos poseen algunas reglas particulares:

- Si el robot se encuentra cargando al niño y existe un camino desde la posición actual del robot hasta el corral pasando solo por casillas vacías, entonces el robot se mueve en esa dirección.
- Analiza en cada turno cual es el objetivo más cercano, Suciedad o Niño, y se mueven en consecuencia, priorizando al niño en caso de empate.
- Si no es posible llevar al niño hasta el corral, ya sea porque el camino está bloqueado por obstáculos, suciedad, o incluso otros niños o robots, entonces obvia al niño, o lo suelta si es que lo estaba cargando, y decide moverse a la suciedad más cercana.
- Si el robot se encuentra en una casilla con suciedad, entonces la limpia antes de continuar su camino.
- Si no existe ningún objetivo viable, entonces el robot no se mueve.
- No tiene en cuenta el proceso de cambio aleatorio del ambiente.
- El robot se mueve dos pasos si está cargando a un niño.
- Posee tres estados, cargando niño, limpiando suciedad o libre.

Módulo *StateRobot.hs*

Este módulo posee la lógica para el funcionamiento de los robots deductivos. Estos poseen la mayoría de los comportamientos de los robots reactivos, solo se diferencian estos en que una vez que fijan un objetivo, guardan su posición hasta que este es alcanzado o ocurre algún cambio del ambiente que imposibilite su cumplimiento. Además, estos robots tienen en cuenta el cambio aleatorio del ambiente, es decir, solo fijan como objetivo a un niño si es posible llegar hasta la posición del mismo y transportarlo hasta el corral en una cantidad de pasos menor que los que restan para el cambio del ambiente, si esto no se cumple entonces se centran en una suciedad.

3. Observaciones sobre la implementaciín.

Se utilizó el paradigma de ¨funciones puras¨ de haskell siempre que se pudo, aprovechando de esta forma las optimizaciones que hace el compilador del lenguaje al utilizar estos elementos. Solo las funciones del ciclo general del programa, y aquellas otras que interactúan de alguna forma con el módulo Random fueron implementadas de forma imperativa, al ser esto absolutamente necesario para su funcionamiento.

El programa se encuentra modularizado, facilitando su edición y comprensión, y permitiendo además que los distintos factores que interactúan con el ambiente (Niños y Robots), queden separados entre si.

Para probar el programa es necesario correr la función main del módulo del mismo nombre, preferiblemente con el compilador de haskell ghci, al hacerlo se obtendrá un log bastante detallado de cada uno de los cambios del ambiente, así como un resumen al final de la ejecución.

4. Modelos de agentes utilizados.

Se utilizó un modelo de agentes puramente reactivo para la implementación de los ReactiveRobots, estos agente evalúan en cada paso el ambiente y toman la solución más adecuada en cada momento siguiendo un conjunto de reglas básicas.

Se utilizó un modelo de agentes deductivos para la implementación de los StateRobots, estos evalúan el ambiente para encontrar un objetivo, y se centran en este hasta su cumplimiento o hasta que algún cambio del ambiente se lo impida. Además, los StateRobots solo fijan un niño si la cantidad de pasos para recoger al niño y devolverlo al corral es menor que la cantidad de turnos hasta el próximo cambio aleatorio del sistema.

5. Experimentación y resultados.

Se experimentó con un ambiente de tamaño 20x20, dos robots, con distintas cantidades de niños y valores para el cambio aleatorio del ambiente. Tal y como se muestra a continuación.

Máximo porciento de suciedad

•	20	30	40	50	60	80	100
4	6	9	11	12	7	5	8
9	32	22	28	22	26	20	20
16	71	63	62	53	59	42	59
25	87	86	87	81	72	63	63
36	91	92	86	82	83	87	83

Robots reactivos: 2, Columnas: 20, Filas: 20, Obstáculos 60

Cantidad de pasos para limpiar el ambiente

•	20	30	40	50	60	80	100
4	147	179	237	216	165	129	149
9	640	481	563	394	589	268	346
16	2009	2210	1485	1056	1006	773	773
25	3655	3424	3022	1714	1421	797	1122
36	5891	3092	2154	1956	1869	1653	1492

Robots reactivos: 2, Columnas: 20, Filas: 20, Obstáculos 60

Máximo porciento de suciedad

•	20	30	40	50	60	80	100
4	17	9	12	5	6	4	4
9	64	43	40	29	22	20	15
16	77	74	55	61	69	53	51
25	89	85	78	75	73	68	71
36	92	91	88	84	82	86	79

Robots deductivos: 2, Columnas: 20, Filas: 20, Obstáculos 60

Cantidad de pasos para limpiar el ambiente

•	20	30	40	50	60	80	100
4	526	198	341	126	151	105	146
9	3375	1043	2232	706	329	391	317
16	9104	3777	1550	1263	1476	1060	1193
25	8184	5409	3589	2233	2416	1457	1005
36	7284	5052	3196	3066	2328	1330	1583

Robots deductivos: 2, Columnas: 20, Filas: 20, Obstáculos 60

Como se puede observar los agentes reactivos se comportan de una mejor manera cuando el intervalo de cambio aleatorio del ambiente es pequeño, esto se debe a que estos agentes no se paran a pensar si es posible alcanzar un niño o no, y realizan siempre la mejor opción en cada turno. Mientras tanto, para valores altos del intervalo de cambio del ambiente, los agentes deductivos logran comportarse igual o algunas veces mejor que los agentes reactivos, sobre todo en las ocasiones en que los agentes deductivos logran tomar una ventaja temprana guardando niños en el corral mientras el tablero del ambiente se encuentra libre de suciedad, cosa que los agentes reactivos les cuesta más trabajo hacer pues estos se entretienen más fácilmente con la suciedad.

Para mejorar estos modelos de agente se pudiera implementar un modelo multiagente, aprovechando que los robots deductivos llevan constancia de cual es su objetivo actual, y permitiendo a los agentes reconsiderar su objetivo si hay algún otro que posea el mismo. Además, se puede agregar ua nueva regla a los agentes, haciendo que se centren en la suciedad si la proporción de esta se encuentra por encima de un umbral, aunque esto puede traer también que el agente empeore su comportamiento si el índice de velocidad de creación de suciedad es mayor que el de limpieza del agente, pues estos solo se centrarían en la limpieza.

Los experimentos aquí recopilados fueron hechos con dos agentes de cada modelo, incluso en los casos en que habían más de 15 niños, lo cual es insuficiente para estas cantidades, en un futuro se podría experimentar con múltiples cantidades de agentes, analizando y estimando las cantidades más adecuadas en cada caso.

Se decidió no mostrar los resultados de los tiempos de ejecución puesto que los agentes deductivos son evidentemente más lentos pues requieren mayor cantidad de operaciones para encontrar y considerar su objetivo.

6. Repo en GitHub

https://github.com/mavaldivie/Tile-World