Proyecto Simulación-Programación Declarativa

Autor: Victor Manuel Cardentey Fundora C411

Principales ideas seguidas para las solución del problema:

El ambiente planteado en el problema es una cuadrícula de $N \times M$ donde cada celda de la cuadrícula puede o no contener distintos elementos presentes en la simulación, los elementos presentes en la simulación son obstáculos, basura, corrales, niños y robots.

El ambiente puede variar de forma natural todos los turnos o de forma aleatoria cada t turnos. La variación natural del ambiente se define como el movimiento de los niños en la habitación, cada niño escoge de forma aleatoria una casilla adyacente (vacía u ocupada por un obstáculo) a la cual moverse en cada turno. La variación aleatoria del ambiente se define como la generación de suciedad, las posiciones de esta suciedad vienen determinadas por la cuadrícula de 3×3 centrada en cada niño, las casillas vacías de esta cuadrícula se convierte en un conjunto de posibles posiciones para la suciedad, luego este conjunto es muestrado con probabilidad uniforme utilizando una cantidad de muestras determinada por la cantidad de niños presentes en la cuadrícula.

Los agentes que intervienen en la simulación son robots, estos se han implementado como agentes reactivos siguiendo la arquitectura de Brooks (*arquitectura de inclusión*). Para ello procedemos a definir las posibles percepciones del agente:

- Detectar que existe un niño fuera del corral (*detect kid not in playpen*)
- Detectar que existe una celda sucia (*detect dirty cell*)
- Detectar si está cargando a un niño (*carrying kid*)
- Detectar si está en una casilla sucia (*at dirty cell*)
- Detectar si está en la misma casilla que un niño (*at kid cell*)
- Detectar si está en sobre el corral (*at playpen*)

Luego las posibles acciones del agente son:

- Moverse hacia un niño (*move to kid*)
- Moverse hacia una celda sucia (*move to dirty cell*)
- Moverse hacia el corral (*move to playpen*)
- Cargar al niño (*pick up kid*)
- Dejar al niño en el suelo (*drop kid*)
- Limpiar la celda (*clean cell*)

Modelos de agentes considerados

Entonces podemos definir los módulos <*perception*, *action*> del agente en el siguiente orden de prioridad de acuerdo a lo descrito en la arquitectura de Brooks:

El primer modelo pensado es bastante simple donde el robot simplemente se acerca al elemento con el que puede interactuar (niño o suciedad) más cercano a él e iteractúa con él de la forma adecuada

- <*carrying kid* and not *at playpen*, *move to playpen*>
- <*carrying kid* and *at playpen*, *drop kid*>
- <*at dirty cell*, *clean cell*>
- <*detect nearest element*, *go to element*>
- <*anything else*, *stay at position*>

El segundo modelo sigue una idea bastante intuitiva que es primero encerrar a los niños en el corral para que no pueda producirse más suciedad ya que tener suciedad en el suelo ayuda a limitar el movimiento de los propios niños, una vez que los niños estén encerrados limpiar la habitación.

- <*carrying kid* and not *at playpen*, *move to playpen*>
- <*carrying kid* and *at playpen*, *drop kid*>
- <*detect kid not in playpen*, *move to kid*>
- <*at dirty cell*, *clean cell*>
- <*detect dirty cell*, *move to dirty cell*>
- <*anything else*, *stay at position*>

Habiendo definido los elementos que intervienen en la simulación procedemos a determinar los objetivos de la simulación, en este caso simularemos el escenario hasta un máximo de T turnos y tendremos la condición de parada de haber alcanzado un 60% de celdas limpias.

Para proveer acceso al estado de la simulación se utiliza un sistema de log el cual recopila la información que se genera la simulación.

Ideas seguidas para la implementación

Primeramente debemos definir como representar el ambiente de la simulación, en este caso la habitación `Room`, se decidió representar el ambiente como una tupla de conjuntos que representaban los distintos elementos presentes en la simulación: `Empty`, `Obstacle`, `Dirt`, `Playpen`, `Kid`, `Robot` y `CRobot`. Cada uno de estos conjuntos tiene como elementos posiciones (x, y) lo cual representa que un elemento del tipo del conjunto esta en dicha posición. El tipo `Room` es una tupla formada por estos conjuntos $< E, O, D, P, K, R, C >$. La elección de este tipo de representación viene dada por el costo de obtener que casillas pertenecen a cada conjunto, esta operación en un diccionario el cual fue la otra posible representación considerada en Haskell es $\mathrm{O}(n^2)$ mientras que al tener los conjuntos ya separados podemos obtener esta información en tiempo constante sin afectarse el tiempo de búsqueda de los elementos dado que tanto `Set` como `Map` en implementan estas operaciones en $\mathrm{O}(\log n)$.

> Nota: El tipo `CRobot` fue utilizado para resolver un caso extremo donde no se podía decidir si un niño estaba en la misma casilla que un niño porque lo estaba transportando o porque lo había soltado en dicha casilla el turno anterior

A continuación debemos considerar el cambio natural del ambiente el cual es el movimiento de los niños, utilizando nuestra representación se puede definir como una operación de conjuntos

> `foreach` $k \in K$ `select random from` $(adyacents(k) \cap E) \cup (adyacents(k) \cap O)$

Luego el cambio aleatorio se realiza de forma similar, sin embargo ahora seleccionanamos n muestras aleatorias en dependencia de la cantidad de niños de la cuadrícula

> `foreach` $k \in K$ `select` n `randoms from` $grid(k) \cap E$ `where` $n = f(|grid(k) \cap K|)$

Teniendo el ambiente definido debemos implementar los agentes, de forma similar a la implementación provista en el libro de texto pág 77, debido a la elección de Haskell se implementó el módulo de decisión utilizando *pattern matching*

```
robotAction room robot logger
    | carryKid room robot = leaveInPlaypen room robot logger
    | existFreeKids room = persuitNearestKid room robot logger
    | robotInDirt room robot = cleanDirt room robot logger
    | isDirtyRoom room = persuitNearestDirt room robot logger
    | otherwise = dontMove room robot logger

robotAction' room robot logger
    | carryKid room robot = leaveInPlaypen room robot logger
    | robotInDirt room robot = cleanDirt room robot logger
    | otherwise = persuitNearestTarget room robot logger
```

Para implementar los sensores del agente, en este caso para determinar la ruta del robot, nos valimos de una implementación de BFS para obtener siempre aquellos elementos más cercanos. Los demás sensores se pueden implementar de forma eficiente con operaciones sobre conjuntos, determinar si el robot está en una casilla vacía es comprobar si la casilla del robot pertenece al conjunto E, mover elementos es una operación de eliminación y otra de inserción sobre el mismo conjunto. Debido al tiempo logarítmico de las operaciones de los conjuntos la mayoría de estas operaciones son bastante eficientes.

Otra consideración necesaria fue respecto a la generación de números aleatorios en Haskell, debido a que en Haskell las funciones son puras estas no deben de tener fuentes internas de entropía por lo que se decidió optar por el enfoque de que las funciones que debían utilizar números aleatorios recibieran como parámetro un generador de números aleatorios y devolvieran a su vez el generador a utilizar para generar el próximo número aleatorio de la secuencia. Esto tiene como consecuencia que dada la misma semilla inicial se produce la misma ejecución lo cual hace que para cambiar el comportamiento de la ejecución se deba cambiar la semilla, por otro lado también hace más fácil estudiar casos específicos dado que los casos de prueba son deterministas y podrían reproducirse tanto como se necesite para su estudio.

Consideraciones obtenidas a partir de la ejecución

Luego de ejecutar varios experimentos se arribó a la conclusión que el segundo modelo (aquel que encierra primero a los niños) se desempeña mejor sin embargo existen limitaciones importantes como la falta de coordinación entre los distintos robots lo cual hace que en ocasiones atrapar a un solo niño que no para de moverse sea una tarea problemática por lo que implementar un mecanismo de coordinación para los agentes sería una buena mejora para el modelo reactivo. Sin embargo el comportamiento del tiempo es irregular incluso para el mismo número de robots y niños dado que la disposición de los elementos dentro de la habitación tiene un gran impacto en el desempeño de los robots.

Enlace al repositorio

Leer el README.md en el repositorio para instrucciones sobre como ejecutar

https://github.com/Vitico99/sim-final-project.git