UNR - FCEIA

ANÁLISIS DE LENGUAJES DE PROGRAMACIÓN

EDSL
Autómatas celulares

Spoletini Bruno

AUTÓMATAS CELULARES

Spoletini Bruno 18 de diciembre de 2024

1. Introducción

1.1. Descripción del proyecto

Este proyecto consiste en un DSL (Domain Specific Lenguaje) embebido en Haskell, el cual permite definir distintos tipos de células, ubicarlas en una cuardrícula, ejecutar una función de transición y visualizar la simulación a través de una interfaz gráfica de usuario (GUI).
El proyecto se divide en dos etapas, la definición de los AC y la visualización de los mismos.

1.1.1. Definición de Autómatas Celulares

Cada Autómata Celular (AC) se define como un conjunto de Células, y un estado inicial que consiste en la ubicación de dichas células en una grilla, junto con una cantidad finita de pasos de transición.
Las células a su vez se definen con un nombre y un color que las identifique, junto con sus reglas de transición $[Birth]/[Survive]$. Estas reglas se basan en que una célula puede aparecer en la grilla en caso de que tenga n celulas vecinas de su mismo tipo, con $n \in [Birth]$, y que cada célula se mantiene en la grilla solo si tiene m células vecinas de su mismo tipo, con $m \in [Survive]$.
Para determinar que célula es vecina de otra se utilizó la Vecindad de Moore, que consiste en las 8 células que la rodean.

1.1.2. Visualización

Para mostrar la simulación a lo largo del tiempo se utilizó el framework Threepenny-gui, el cual genera un servidor web que puede ser accedido a través de un navegador para visualizar la interfaz gráfica, en conjunto con FRP (Functional Reactive Programming) para permitir una interacción fluida con dicha interfaz.
La misma ofrece:

- Grilla interactiva
- Selector de los AC definidos
- Botonera para controlar los pasos de transición
- Selector de Células para los AC multi-celulares
- Posibilidad de exportar el estado del AC actual
- Consola con el historial de comandos

2. Manual de uso e instalacion de software

2.1. Instalación

Para utilizar el DSL de autómatas celulares, es necesario tener instalado Haskell y Stack. Para instalar el proyecto se debe clonar el repositorio, acceder a la carpeta, e instalar las dependencias con stack:

```
git clone https://github.com/BrunoSpoletini/CellularAutomataEDSL
cd CellularAutomataEDSL
stack build
```

2.2. Ejecución

Para ejecutar el proyecto, situado en la carpeta del mismo, debe ejecutarse el comando:

```
stack run
```

Esto realizará la compilación de todos los archivos que se encuentren en la carpeta "/definitions" e iniciará la GUI con dichos autómatas disponibles para ser visualizados.

Para acceder a la interfaz se deberá ingresar por medio de un navegador web a la IP 127.0.0.1:8023.

2.3. Creación de Automatas Celulares

Para definir un nuevo AC se deberá crear un archivo de texto en la carpeta "/definitions" siguiendo la gramática dada por (2.4). Cada autómata deberá tener al menos una célula, y a su vez, cada célula debe contar con un nombre único, un color válido (4.2) y un set de reglas de transición.

Existen 4 comandos para interactuar con el estado del autómata en el archivo de definición:

- $DEFCELL\, cell_{name} = (cell_{color}, [rule_{birth}], [rule_{survive}])$: define una célula cuyo nombre se usará para posicionarla en la grilla, su color se mostrará en la interfaz gráfica, y sus pasos de transición estarán dados por sus reglas de nacimiento y supervivencia.
- $UPDATECELL\ (x,\ y)\ cell_{name}$: actualiza la posición $(x,\ y)$ de la grilla con la célula $cell_{name}$. En caso de que dicha célula ya se encuentre en esa posición, la elimina.
- $STEP$: realiza un paso de transición
- $STEPS\ n$: realiza n pasos de transición

2.3.1. Ejemplo

Para definir el autómata Game of Life con un oscilador como estado inicial, creamos un archivo "/definitions/conway.txt" que contenga:

```
DEFCELL conway = (Black, [3], [2,3])
UPDATE (7,4) conway
UPDATE (7,5) conway
UPDATE (7,6) conway
```

Por defecto se visualizará el AC definido en “default.txt”.

2.4. Gramática

Gramática utilizada para formar las expresiones del archivo de definición:

Comm ::= *DefCell*
| ’UPDATE’ *Position Var*
| ’STEP’
| ’STEPS’ *Number*

DefCell ::= ’DEFCELL’ *Var* ’=’ ’(’ *Var* ’,’ ’[’ *NList* ’]’ ’,’ ’[’ *NList* ’]’ ’)’

NList ::= *Number NList*
| ’,’ *NList*
| ϵ

Position ::= ’(’ Number ’,’ Number ’)’

Number ::= *Digit*
| *Digit Number*

Digit ::= ’0’ | ’1’ | ’2’ | ’3’ | ’4’ | ’5’ | ’6’ | ’7’ | ’8’ | ’9’

Var ::= *Letter* | *Letter Var*

Letter ::= ’A’ | ’B’ | ... | ’Z’

3. Organizacion de los archivos

El proyecto está organizado en varios módulos para facilitar su mantenimiento y comprensión. A continuación, se describe la estructura del proyecto:

- **Main.hs**: Se encarga de compilar los archivos de entrada, hacer el manejo de errores e iniciar la UI.

- **Automata.hs**: Define las funciones de la mónada que lleva el estado del AC y sus operaciones correspondientes.
- **BehaviourManager.hs**: Genera y gestiona los eventos de interacción con la UI.
- **Common.hs**: Declara los tipos de datos que se usan a lo largo del proyecto
- **Components.hs**: Genera los componentes de la UI.
- **Config.hs**: Declara constantes usadas por la UI.
- **Front.hs**: Genera la estructura de la UI y gestiona su actualización.
- **Monads.hs**: Define las clases de las mónadas MonadState y MonadError
- **parse.y**: Archivo generador del parser "parse.hs"
- **PrettyPrint.hs**: Pretty printer encargado de mostrar los comandos por la consola de la UI.

4. Decisiones de diseño

4.1. Estructuras utilizadas

El tipo de dato Comm es el que se realizan las interacciones con el entorno del AC, y se divide en dos partes:

- Comandos de entrada: los primeros cuatro comandos se utilizan para la definición del autómata según se explicó anteriormente.
- Comandos de la UI: se ejecutan al interactuar con la UI, y se describen a continuación:
  - Restart Env: Se ejecuta al elegir un autómata o al clickear en el boton Reset. Actualiza el estado de la monada con el enviroment del AC elegido.
    Esto podria hacerse generando un nuevo autómata y re-compilando el archivo del AC elegido, pero este comando existe por cuestiones de eficiencia.
  - Select Ident: Se ejecuta al seleccionar una célula de la lista, y actualiza la célula elegida en el environment.
  - UpdatePos Pos: Se ejecuta al hacer click en el canvas. Este click mapea la posición en la cuál se clickeó con una célula de la grid, y actualiza su valor en el entorno según la célula que se encuentre seleccionada.

```
-- Comandos interactivos
data Comm =   DefCell Variable Variable [Int] [Int]
            | UpdateCell Pos Variable
            | Step
            | Steps Int
```

```
    -- Comandos de la UI
    | Restart Env
    | Select Ident
    | UpdatePos Pos
```

El estado del autómata, a su vez, está compuesto por los datos de la grilla, un array con las células definidas, y la célula seleccionada por la UI.

```
type Env = (GridData, ([CellData], CellData))
```

Por último, los datos de la grilla se componen de la grilla en sí y un array que representa las posiciones de las células que fueron modificadas en la última actualización del entorno,

```
data GridData = GridData {  grid :: Grid,
                            changes :: [Pos] }
```

4.2. Colores permitidos

Los colores que se usan para definir las células se leen como una String al pasar por el parser. Al momento de agregar dicha célula a la mónada de estado, se verifica si el color ingresado puede ser transformado en el tipo de dato color, definido como:

```
data Color = Black | Silver | Gray | Grey | White | Maroon | Red | Purple
| Fuchsia | Green | Lime | Olive | Yellow | Navy | Blue | Teal | Aqua
| Brown | AliceBlue
```

Se decidió por esta forma de interpretar el color, ya que resulta más intuitivo que describirlo según sus valores RGB.

4.3. Pretty Printer

La visualización del estado del AC en la UI consiste en la ejecución de los comandos de definición del autómata, combinados con los comandos ejecutados por la UI. Pero para poder exportar el estado actual a un archivo, los comandos de la UI deben ser “traducidos” a comandos de definición. Aquí entra en juego el Pretty Printer para realizar dicha traducción.
La UI muestra una consola, que contiene los comandos necesarios para replicar el estado del AC que se muestra por pantalla en cada momento. Dichos comandos pueden exportarse a través de un boton a un archivo “out.txt”.

5. Fuentes consultadas

https://wiki.haskell.org/index.php?title=Functional_Reactive_Programming
https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life
https://en.wikipedia.org/wiki/Moore_neighborhood

https://hackage.haskell.org/package/threepenny-gui-0.9.4.1/docs/Graphics-UI-Threepenny.html

https://conwaylife.com/wiki/Polystate_Life

https://github.com/thma/ThreepennyElectron/blob/master/README.md