DSL: Automatas Celulares

Descripcion

Idea general

El presente lenguaje de dominio especifico pretende permitir indicar el comportamiento de automatas celulares, asi como tambien observar su evolucion a partir de un estado inicial.

La motivacion principal es proveer una manera sencilla y general de estudiarlos, sin necesidad de tener conocimientos previos de programacion.

Alcances

Es posible especificar automatas en terminos de vecindad de Moore y vecindad de von Neumann, con funciones de transicion que pueden contar y comprar elementos de su vecindad.

Dependencias

Para poder compilar el programa debera contar con un compilador de *Haskell* y las libreria *UI.NCurses* y *Data.Vector*. Si utiliza *Ubuntu*, puede instalar el compilador *GHC* con la instruccion:

```
sudo apt install ghc
```

Para instalar la libreria *UI.NCurses* necesitara instalar previamente los paquetes *c2hs* y *cabal-install* para lo cual en *Ubuntu* puede utilizar el siguiente comando:

```
sudo apt install cabal-install c2hs
```

A continuacion bastara con ejecutar:

```
sudo cabal install ncurses vector
```

Manual de uso

Instrucciones de compilacion

Generacion de parser

Si por alguna razon desea modificar el modulo de parseo, debera contar con la herramienta de generacion de parsers *Happy*. Para instalar dicha herramienta en *Ubuntu* puede utilizar la instruccion:

```
sudo cabal install happy
```

Una vez instalada, puede modificar el archivo *Parser.y* y luego generar el parser mediante la instruccion:

```
happy Parser.y
```

Programa principal

Para compilar el programa basta escribir la orden:

```
ghc Main.hs -O2
```

El uso de la opcion *-O2* no es obligatorio, sin embargo el impacto en la perfomance es notorio.

Instrucciones de ejecucion

Linea de comandos

Si dispone del programa ya compilado puede utilizar la siguiente instruccion para correr el programa:

```
./Main AUTOMATA INICIAL [FRONTIER START FRAMES SKIP TIME]
```

donde *AUTOMATA* es el archivo de definicion del automata y *INICIAL* es el estado inicial. De lo contrario puede ejecutar el programa de la siguiente manera:

```
runhaskell Main.hs AUTOMATA INICIAL [FRONTIER START FRAMES SKIP TIME]
```

Sin embargo si opta por esta manera, la perfomance sera notablemente mas baja.

Opciones

- *FRONTIER*: Determina el tipo de frontera del automata. Valor por defecto: *Wrap*. Las opciones disponibles son:
  - *Wrap*: Se considera al universo como si sus extremos se tocaran.
  - *Reflect*: Se considera que las celulas fuera del univers reflejan los valores de aquellas dentro del universo.
  - *"Open" 'c'"*: Se considera que las celulas fuera del universo tienen todas el valor fijo *c*.
- *START*: Numero de generacion inicial del automata. Valor por defecto: 0.
- *FRAMES*: Numero de generaciones totales del automata. Valor por defecto: 10900.
- *SKIP*: Salto entre generaciones. Valor por defecto: 0.
- *TIME*: Milisegundos de pausa entre generaciones. Valor por defecto: 0.

Manejo del programa

Para manipular la animacion del automata, puede utilizar las siguientes combinaciones de teclas durante la ejecucion del programa:

- *Barra espaciadora*: Pausa o reanuda la animacion.
- *Flecha derecha*: Avanza la animacion un solo paso.
- *CTRL+C*: Sale del programa.

Definicion de automatas

Descripcion

Definicion de representaciones

La definicion de un automata se divide en dos partes: la primera indica como se representan los diferentes estados en pantalla; y la segunda define el comportamiento del automata.

Para definir que el estado e debe representarse como el caracter c con color de fondo x y color principal y, la linea que debe escribirse es:

```
'e': 'c' x y
```

Si para algun estado no se define una representacion, por defecto se utilizara el mismo caracter que representa el estado con fondo negro y color blanco.

Definicion de reglas

Luego de definir las diferentes representaciones en lineas sucesivas, a continuacion se deben definir las reglas que definen el comportamiento del automata. Para indicar que estando en el estado a se debe pasar en cualquier caso al estado b debe escribirse:

```
State == 'a': 'b'
```

Si ademas de estar en el estado a debe cumplirse las condiciones $c1$, ..., cn para poder pasar al estado b debera escribirse:

```
State == 'a' && c1 && ... && cn: 'b'
```

donde las condiciones pueden ser las siguientes:

- `Chebyshev('x',i) ?? j`: Verifica la relacion entre la cantidad de estados `x` que se encuentran a distancia Chebyshev `i` del estado actual; y `j`.
- `Manhattan('x',i) ?? j`: Analogamente para distancia Manhattan.
- `North(i) ?? 'x'`: Verifica la relacion entre el estado que se encuentra `i` celdas hacia arriba del estado actual, y `x`.

- `South(i) ?? 'x'`: Analogamente para el estado que se encuentra `i` celdas hacia abajo.
- Tambien pueden utilizarse de forma similar, los puntos cardinales `East`, `West`, `NE`, `NW`, `SE` y `SW`.

y `??` es uno de los siguientes operadores de comparacion: `==`, `<=`, `>=`, `<`, `>`, `!=`.

Las reglas se intentaran en el orden definido. Si una regla debe aplicarse, las siguientes se omitiran; y si ninguna regla puede aplicarse el estado permanecera intacto.

Gramatica

La siguiente gramatica define formalmente el lenguaje:

```
AUTOMATA ::= STATES
             RULES

STATES ::= Lambda
         | 'CHAR': 'CHAR' COLOR COLOR STATES

COLOR ::= Black | Red | Green | Yellow | Blue | Magenta | Cyan | White

RULES ::= Lambda
        | State == 'CHAR' && COMPARISON: 'CHAR' RULES
        | State == 'CHAR': 'CHAR' RULES

COMPARISON ::= DISTANCE('CHAR',INT) CMP INT && COMPARISON
             | CARDINAL(INT) CMP 'CHAR' && COMPARISON
             | DISTANCE('CHAR',INT) CMP INT
             | CARDINAL(INT) CMP 'CHAR'

CMP ::= == | <= | >= | < | > | !=

DISTANCE ::= Chebyshev | Manhattan

CARDINAL ::= North | South | East | West | NE | NW | SE | SW
```

Ejemplos

La carpeta *examples* contiene definiciones de automatas populares que sirven de ejemplo para comprender el uso del lenguaje.

Informacion adicional

Distribucion de modulos

Con el fin de facilitar el mantenimiento y lectura del codigo fuente, se ha decidido separar el codigo en los siguientes modulos:

- *Main.hs*: Es el modulo principal del programa. Se encarga principalmente de mostrar la animacion del automata por pantalla.
- *Matrix.hs*: Definiciones de funciones para el manejo de matrices.
- *Parser.y*: Definicion de la gramatica para el parseo de los automatas.
- *Rules.hs*: Definicion de funciones que calculan el funcionamiento de los automatas.
- *Types.hs*: Definiciones de los tipos utilizados en el programa.

Desiciones de diseño

Libreria *UI.NCurses*

Originalmente para mostrar el estado de un autiomata en pantalla se utilizaba la funcion `putStrLn` seguido de `clearScreen`. Sin embargo esto producia perdida de rendimiento y parpadeo en la pantalla.

La libreria *UI.NCurses* no borra la pantalla, sino que redibuja sobre el estado anterior, y ademas solo lo hace en las partes de la pantalla que hayan cambiado. Esto soluciona ambos problemas.

Libreria *Data.Vector*

La perfomance ofrecida por las listas secuenciales de Haskell no son adecuadas para este programa, pues es necesario acceder a los diferentes elementos de un estado de manera eficiente.

El acceso en tiempo constante ofrecido por esta libreria es una solucion a los problemas presentados en la implementacion original.

Extension *XRankNTypes*

Las funciones de comparacion tienen tipo `Int -> Int -> Bool` o `Char -> Char -> Bool`. Podemos generalizar esto como `Ord a => a -> a -> Bool`.

Por una cuestion de diseño del generador de parsers *Happy*, esto no era posible. Inicialmente se opto por definir un nuevo tipo de datos que era manipulado por la funcion de observacion, sin embargo el uso de esta extension del lenguaje permitio implementar la idea original, logrando un codigo mas claro.

Bibliografia

Para mayor conocimiento, puede consultar los siguientes articulos de *Wikipedia* que sirvieron de ayuda e inspiracion en este trabajo:

- Automata Celular
- Vecindad de Moore
- Vecindad de von Neumann
- Distancia de Chebyshov
- Geometria del taxista
- Juego de la Vida
- Hormiga de Langton
- Seeds
- Brian's Brain
- Wireworld

Tambien contribuyeron a la confeccion de este programa, la documentacion de Happy y de UI.NCurses, asi como tambien el libro Learn You a Haskell for Great Good! de Miran Lipovača y el material de estudio de la catedra de Analisis de Lenguajes de Programacion de la Facultad de Ciencias Exactas, Ingenieria y Agrimensura provisto por los profesores Mauro Jaskelioff, Cecilia Manzino, Aldana Ramirez y Juan Manuel Rabasedas.

Acerca del autor

Mi nombre es Damian Ariel, soy estudiante del tercer año de la carrera de Licenciatura en Ciencias de la Computacion en la Facultad de Ciencias Exactas, Ingenieria y Agrimensura de la ciudad de Rosario.

Este trabajo fue presentado en el marco de la catedra de Analisis de Lenguajes de Programacion.