Sistema de Lindermayer

Carlos Gomez

30 de marzo de 2011

1. Introducción

El objetivo de este proyecto es extender un proyecto ya existente de la Comunidad Haskell San Simón.

Trabajaremos sobre el proyecto Sistema-L: http://devel.comunidadhaskell.org/l-system

El proyecto Sistema-L es un graficador de gramáticas 'l-system'[1], puedes encontrar mas información del tema en:

Libro The Algorithmic Beauty of Plants (existe una version disponible en la web)

Wikipedia http://en.wikipedia.org/wiki/L-system

Comenzaremos haciendo una pequeña descripción del proyecto, para luego especificar las tareas a realizar.

2. Descripción

El proyecto Sistema-L interpreta estructuras simples de gramáticas del Sistema-L. Las estructuras del Sistema-L estan representadas en Haskell a través de una tupla de 3 elementos:

```
type L = (V, W, P)

type V = [String]
type W = String
type P = [(Char, String)]
```

donde 'V' es el alfabeto del sistema, 'W' representa el símbolo de inicio y 'P' representan las producciones.

La idea básica del proyecto es dibujar una secuencia de movimientos de tortuga. Donde la secuencia es el resultado de derivar n-veces la gramática del Sistema-L.

Por ejemplo, si tenemos la estructura:

[1] Sistema de Lindermayer

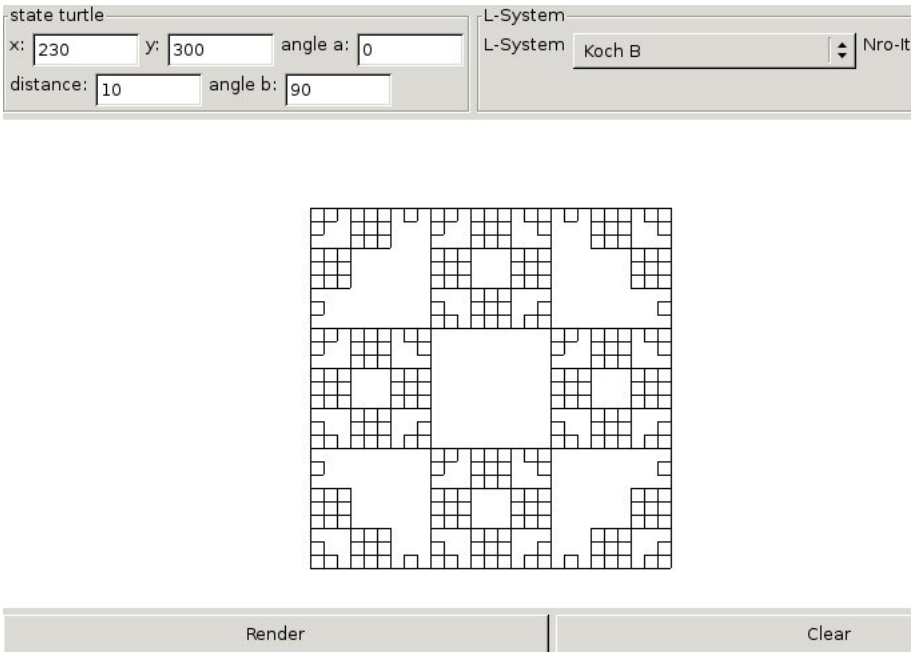
$kochb = ([\text{"F"}], \text{"F–F–F–F"}, [('F', \text{"FF–F–F–F–FF"})])$

podemos derivarla 3 veces, y obtenemos la siguiente secuencia para dibujar:

```
*Parser> generate kochb 3
"FF-F-F-F-FFFF-F-F-F-FF-FF-F-F-F-FF-FF-F-F-F-FF-FF-F-F-F-FF-FF-F-F-F-
FFFF-F-F-F-FFFF-F-F-F-FFFF-F-F-F-FF-FF-F-F-F-FF-FF-F-F-F-FF-FF-F-F-F-
FF-FF-F-F-F-FFFF-F-F-F-FF-FF-F-F-F-FFFF-F-F-F-FF-FF-F-F-F-FF-FF-F-F-
F-FF-FF-F-F-F-FF-FF-F-F-F-FFFF-F-F-F-FF-FF-F-F-F-FFFF-F-F-F-FF-FF-F-
F-F-FF-FF-F-F-F-FF-FF-F-F-F-FF-FF-F-F-F-FFFF-F-F-F-FF-FF-F-F-F-FFFF-
F-F-F-FF-FF-F-F-F-FF-FF-F-F-F-FF-FF-F-F-F-FF-FF-F-F-F-FFFF-F-F-F-FF-
FF-F-F-F-FFFF-F-F-F-FF-FF-F-F-F-FF-FF-F-F-F-FF-FF-F-F-F-FF-FF-F-F-F-
FFFF-F-F-F-FFFF-F-F-F-FFFF-F-F-F-FF-FF-F-F-F-FF-FF-F-F-F-FF-FF-F-F-
F-FF-FF-F-F-F-FFFF-F-F-F-FF-FF-F-F-F-FFFF-F-F-F-FF-FF-F-F-F-FF-FF-F-
F-F-FF-FF-F-F-F-FF-FF-F-F-F-FFFF-F-F-F-FFFF-F-F-F-FFFF-F-F-F-FF-FF-
F-F-F-FF-FF-F-F-F-FF-FF-F-F-F-FF-FF-F-F-F-FFFF-F-F-F-FF-FF-F-F-F-FFFF-
F-F-F-FF-FF-F-F-F-FF-FF-F-F-F-FF-FF-F-F-F-FF-FF-F-F-F-FFFF-F-F-F-FF-FF-F-F-F-FFFF-F-F-F-FF-F
```

La secuencia generada es interpretada por una tortuga, esta tortuga conoce ciertos comandos para dibujar: El comando 'F' le indica que avance una distancia en una dirección dada, y también que debe dibujar una linea por su recorrido. El comando 'f' le dice que haga lo mismo que 'F' pero que no dibuje una linea. El comando '+/−' le dice que incremente/decremente su angulo en una cierta cantidad.

Por ejemplo, el gráfico que resulta de la secuencia generada anteriormente es:



Para una descripción mas detallada del código puede revisar su documentación, o directamente el código.

3. Tarea

1. La primera tarea es escribir una función que verifique si un Sistema-L es correcto.

   Para verificar la corrección de un Sistema-L se puede seguir lo siguiente:

   a) No deben existir elementos repetidos en el alfabeto de un Sistema-L.

   b) La lista de símbolos de inicio no puede ser vacia.

   c) Los símbolos de inicio 'W' deben pertenecer al alfabeto.

   d) Debe existir solo una regla de producción para cada símbolo del alfabeto. Si existe algún símbolo que no esta especificado en el conjunto de producciones se debe generar una producción de identidad para ese símbolo (Por ejemplo, si ese símbolo es 'a', se debe generar '$a->a$').

   e) El símbolo del lado izquierdo de una producción debe pertenecer al alfabeto.

   f) El lado derecho de una producción puede ser vacio o un conjunto de símbolos que pertenecen al alfabeto.

   La función de verificación podría devolver un tipo `Either Errores SistemaL`, en el lado izquierdo los errores, y en el lado derecho el Sistema-L correcto.

2. Escribe un analizador sintáctico (Parser) para un Sistema-L. El analizador sintáctico debe seguir la siguiente gramática:

```
Sistema_L ::= nombre "{" alfabeto inicio producciones "}"
alfabeto ::= "alfabeto" "=" lista_simbolos
inicio ::= "inicio" "=" lista_simbolos
producciones ::= "producciones" "=" "[" lista_prods "]"

lista_simbolos ::= "[" simbolos "]"
simbolos ::= epsilon
  |  simbolo { "," simbolo }*

lista_prods ::= epsilon
 |  produccion { ";" produccion }*

produccion ::= simbolo "->" lista_simbolos

nombre ::= lista de palabras

simbolo ::= alfanumerico y otros simbolos (+-)
```

Un ejemplo para esta gramática podría ser:

```
Koch B {
alfabeto = [F,f,+,-]
inicio   = [F,-,F,-,F,-,F]
producciones = [ F -> [F,+,F]
   ; f -> []
   ]
}
```

Si deseas puedes crearte otro tipo de datos para representar un Sistema-L y reemplazar el que actualmente existe.

3. Modifica el Parser de manera que en un archivo puedan estan descritos uno o mas Sistema-L. El resultado de llamar al parser debe ser una lista de Sistema-L.

4. Ahora debes integrar la función de verificación de Sistema-L con el parser, de manera que, después de procesar la entrada con el parser, se verifique si el Sistema-L es correcto, si no fuera correcto se imprime los errores, y si fuera correcto se devuelve el Sistema-L.

5. Lo siguiente es integrar el nuevo parser + la verificación con el GUI. Para esto, debes crear un archivo que sirva como base de datos de Sistema-L para el GUI. Así, cuando el programa inicie, llame al parser para procesar ese archivo.

   De acuerdo al tipo de dato que hayas escogido, posiblemente tengas que modificar la generación del Parser para la secuencia (función 'generate' de Parser.hs) o tal vez la forma de procesar los comandos (función 'doCommand' en Process.hs).

6. Hasta ahora la tortuga conoce instrucciones sencillas (ejemplo 'F', 'f', '+', '−'), en estas dos siguientes tareas le enseñaremos dos técnicas de dibujado.

   Reescritura de lados[2]. Como mencionamos anteriormente, la tortuga solo conoce ciertos comandos para dibujar lineas y modificar ángulos, además el resultado de las secuencias generadas deben ser comandos que la tortuga conoce. Así, esta técnica permite extender las reglas de generación de secuencias sin modificar los comandos que conoce la tortuga.

   Por ejemplo, podemos tener un símbolo `Fl` que produce `Fl -> Fl+F-`. Para generar su secuencia simplemente remplazamos cada ocurrencia de `Fl`

[2]Edge Rewriting

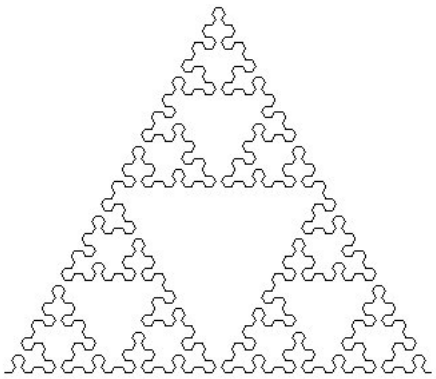
por su lado derecho (así como lo hemos estado haciendo hasta ahora), pero cuando se trate de dibujar el comando `Fl`, la tortuga debe simplemente interpretarlo como si fuera `F`.

Entonces, esta técnica le permite derivar en formas diferentes, pero dibujarlas como una sola forma.

Por ejemplo, esta nueva técnica nos permite dibujar:

```
Sierpinski gasket
    {alfabeto = [Fl,Fr,+,-]
     inicio   = [Fr]
     producciones = [ Fl -> [Fr,+,Fl,+,Fr]
                    ; Fr -> [Fl,-,Fr,-,Fl]
                    ]
    }
}
```

Renderizando con un angulo de 60 y haciendo 6 iteraciones, obtenemos:



7. **Reescritura de Nodos**[3]. Esta técnica nos permite representar partes de una figura en las producciones. Por ejemplo:

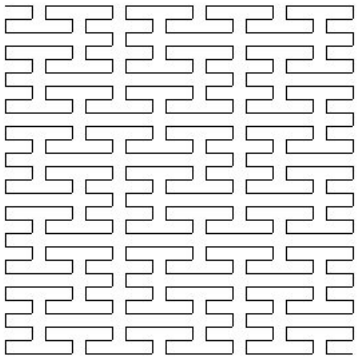
```
3 x 3 Macrotile
    {alfabeto = [L,R,F,+,-]
     inicio   = [L]
     producciones = [ L -> [L,F,R,F,L,-,F,-,R,F,L,F,R,+,F,+,L,F,R,F,L]
                    ; R -> [R,F,L,F,R,+,F,+,L,F,R,F,L,-,F,-,R,F,L,F,R]
                    ]
    }
```

[3]Node Rewriting

En el ejemplo, 'L' y 'R' representan partes de la figura (podría ser cualquier otro nombre, no necesariamente 'L' o 'R'). La técnica nos indica que podemos usar símbolos y producciones para representar partes de una figura, pero cuando se trate de dibujar esas partes podemos hacer una de dos:

- Eliminar todos los símbolos o comandos que no conozca la tortuga.
- O simplemente no hacer nada cuando se encuentre ese tipo de comandos.

Así, si iteramos 3 veces con un angulo de 90 en el anterior ejemplo, obtenemos:



8. Finalmente, añade la posibilidad de dibujar con colores. La idea para esta parte es enseñarle a la tortuga a que reconozca ciertos colores (Rojo, Amarillo, Verde, Azul, ...), estos pueden estar como nombre en los símbolos. Así en cada una de sus ocurrencias se cambia el color para los comandos siguientes.

   Independiente de la manera que lo hagas, debes modificar el estado de la tortuga para que guarde el color actual que esta manejando.

Ejercicios Extras

9. Extiende los comandos de la tortuga para que dibuje en 3 dimensiones. Para mas información, revisa los materiales del inicio del documento.