DM 509 Programming Languages

Fall 2011 Project (Part 2)

Department of Mathematics and Computer Science
University of Southern Denmark

December 2, 2011

Introduction

The purpose of the project for DM509 is to try in practice the use of logic and functional programming for small but non-trivial examples. The project consists of two parts. The first deals with logic programming and the second part with functional programming.

Please make sure to read this entire note before starting your work on this part of the project. Pay close attention to the sections on deadlines, deliverables, and exam rules.

Exam Rules

This second part of the project is a part of the final exam. Both parts of the project have to be passed to pass the course.

Thus, the project must be done individually, and no cooperation is allowed beyond what is explicitly stated in this document.

Deliverables

There is one deliverable for this first part of the project: A short project report in PDF format (2-5 pages without front page and appendix) has to be delivered. This report should contain the following 7 sections:

- front page
- specification
- design
- implementation
- testing
- conclusion
- appendix including all source code

The report has to be delivered in TWO copies:

- 1 electronic copy using Blackboard's Assignment Hand-In functionality
- 1 paper copy (to the teacher's mailbox at the IMADA secretariat)

Deadline

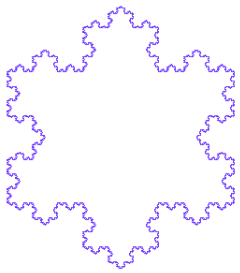
January 13, 2012, 12:00

The Problem

Fractals are geometric objects that are similar to themselves on arbitrarily small scales. There are many examples of fractals in nature, and they form some of the most beautiful structures you can find. One example is snowflakes, but also lightning has a fractal structure. Another example are ferns, where each part is similar to the whole.

Many fractals can be generated by using so-called L-Systems. These systems describe a start state (depth 0) and a set of rules how to evolve a state into a state of larger depth. For more background on L-Systems, have a look at the Wikipedia article:

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



For example, to generate a Koch curve, we start with the initial (depth 0) state `F` signifying a forward move, i.e., a straight line. The rule for expanding to the next depth is given as a replacement rule.

```
F -> F L F R F L F
```

where `L` is a 60 degree turn to the left and `R` is a 120 degree turn to the right. That is, we replace a straight line by a straight line, a left-turn, a straight line, a sharp right-turn, a straight line, a left-turn, and a fourth and final straight line.

Thus, the state for depth 1 is `F L F R F L F`. To get to depth 2, we have to apply the rule again to all positions of the state where it is possible, i.e., we have to replace each of the four `F` by `F L F R F L F`. The result is F L F R F L F L F L F R F L F R F L F R F L F L F L F R F L F where the new sections are underlined to aid your understanding.

To get to depth 3, we would have to replace each of the 16 `F`s by the right side of the rule. For the sake of brevity, I leave this exercise to you.

In general, if there is more than one rule, all rules need to be applied in parallel. That is, all letters, for which there is a rule, need to be replaced by the rules' right-hand sides at the same time to go from depth n to depth $n + 1$. Letters, for which there is rule defined are copied into the new state.

Let us take a look at the file `koch.fdl` available from the project section of the course home page.

```
start F
rule F -> F L F R F L F
length 2
depth 5
cmd F fd
cmd L lt 60
cmd R rt 120
```

The first line gives the start state, i.e., the state `F` for depth 0. The second line give the only rule needed for the Koch curve, i.e., to replace `F` by `F L F R F L F`. The third line specifies the length of each segment, i.e., each straight line will be 2 units long. The fourth line specifies that states should be expanded to depth 5 before drawing the fractal. Finally, the Lines 5 to 7 specify that `F` is a straight line, `L` is a 60 degree left-turn and `R` is a 120 degree right-turn.

Task 0 – Preparation
Download the file `fdl.hs` and load it into hugs using

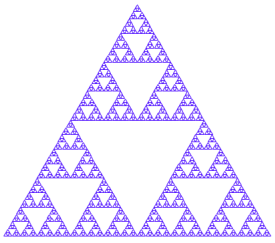
```
hugs fdl.hs
```

Test the functionality using the following command (in one line):

```
runGraphics (withWindow_ "Test" (1000, 600) (\ w -> drawIt
(w, 500, 300, 0, 50) [Forward, RightTurn 90, Forward,
RightTurn 90, Forward, RightTurn 90, Forward] >> getKey w))
```

You should see a window with a white square on a black background.

Task 1 – Applying Rules to a State



Your first task is to write a function `apply :: State -> [Rule] -> State` that takes a state (i.e., a sequence of letters) and applies to each letter of this state the first applicable rule from the list of rules. As specified before, letters that have no applicable rules are copied to the new state.

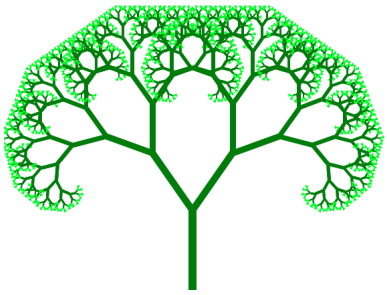
This function corresponds to going from depth n to depth $n + 1$. Test your function by calling it with the following command:

```
apply "FXRYF" [Rule 'X' "XRYF", Rule 'Y' "FXLY"]
```

This should produce the following result:

```
"FXRYFRFXLYF"
```

Task 2 – Going to Target Depth



Your second task is to write a function `expand :: State -> [Rule] -> Int -> State` that takes an initial state, a set of rules, and a target depth. Using your `apply` function, it has to apply the rules repeatedly in parallel until the target depth is reached.

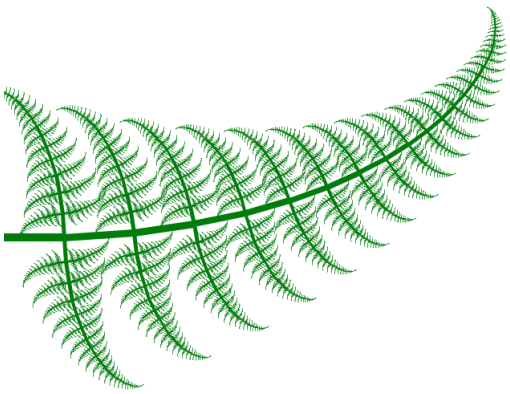
This function corresponds to going from depth 0 to depth n. Test your function by calling it with the following command:

```
expand "FX" [Rule 'X' "XRYF", Rule 'Y' "FXLY"] 2
```

This should again produce the following result:

```
"FXRYFRFXLYF"
```

Task 3 – Processing a Fractal



Your third task is to write a function `process :: Fractal -> [Command]` that takes a value of type `Fractal` and produces a list of commands to draw the fractal. Your function needs to first use `expand` to get the state for the target depth. Then it should use the function of type `Char -> Command` of the fractal to map the state into a list of values of type `Command`.

This function corresponds to translating a fractal description into a list of turtle graphics commands. Test your function by calling it with the following command:

```
main
```

This should produce a white-on-black version of the snowflake as shown on Page 3, i.e., of three Koch curves put together. Also test your code using the other `.fdl` files and include screenshots in your reports.

Task 4* – Support for Line Widths and Colors
The fractals look nice enough, but some colors and wider lines would make them more pretty. Your challenge task is to extend the Fractal Descripton language by the commands `color` and `width` where color gets a color name, a color code or "`random`" as an argument while width gets a float.

In order to accomplish this, you need to read more about the functionality of the Haskell Graphics Library:

`http://hackage.haskell.org/package/HGL`

Here is an example for a nicer dragon curve:

```
start F X
rule X -> X R Y F
rule Y -> F X L Y
length 3
depth 13
color random
width 2.0
cmd F fd
cmd X nop
cmd Y nop
cmd L lt 90
cmd R rt 90
```

Note that this task is optional and does not have to be solved for this part of the project to be considered as passed.