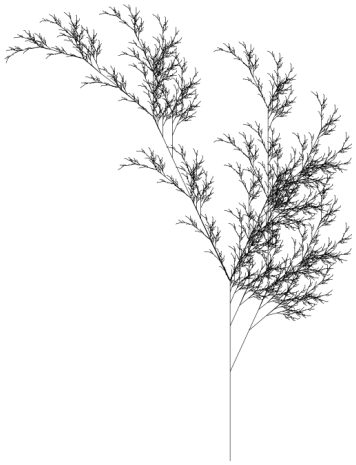
Functional Programming

http://proglang.informatik.uni-freiburg.de/teaching/functional-programming/2019/

Programming project – Draw L-systems in Haskell

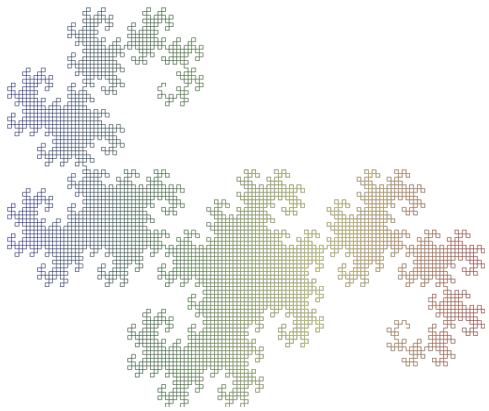
July 3, 2019

Your goal is to create a program to draw L-systems [Wikipedia, 2019b]. L-systems are a form of grammar that is often used for artistic or simulation purposes, notably in biology [Prusinkiewicz and Lindenmayer, 1990]. For instance, the figures below contain two L-systems, one that draws a stylised shrub and another that draws the dragon curve.



```
angle 25
axiom X
X -> F-[[X]+X]+F[+FX]-X
F -> FF
```

Figure 1: L-system for a fern



```
angle 90
axiom X
X -> X+YF+
Y -> -FX-Y
```

Figure 2: L-system for the dragon curve

1 Project

To work on this project you should form groups of two (or exceptionally three) students. The implementation should contain a file `Readme` describing how to compile and use the program. For the final version, each group should send a file "project-FP-<NAMES>.tar" to both Prof. Peter Thiemann and Gabriel Radanne by email. The email should be titled "project FP <NAMES>". The deadline for the project is the **22th July 2019**. Additionally, each group will make a short presentation (10mins) to the rest of the class about their project and its specificities on the 24^{th} July 2019. There will be feedback, but no formal grade for the project and it will not influence the grade of the course. However, one of the exercises of the exam will refer to the project. Additionally, code submitted for the project before the deadline will be accessible during the exam.

The project consists of two parts, a core section that all groups should complete and a set of potential extensions. Each group should pick at least two extensions.

2 Core

The core part of this project consists of three components that must be designed and implemented:

1. a parser for L-systems,
2. an evaluator for L-system,

```
axiom X

X -> X.X
. -> ...
```

```
0: X
1: X.X
2: X.X...X.X
3: X.X...X.X.........X.X...X.X
```

Figure 3: A simple L-system and its first 3 generations

$$
\begin{aligned}
\langle system \rangle &::= \langle header \rangle^* \ \langle rule \rangle + \\
\langle header \rangle &::= \text{`}\texttt{axiom}\text{'} \ \mathcal{I}d \mid \text{`}\texttt{angle}\text{'} \ n \mid \ldots \\
\langle rule \rangle &::= \mathcal{I}d \ \text{`}\texttt{->}\text{'} \ \langle atom \rangle^* \\
\langle atom \rangle &::= \mathcal{S}ymb \mid \mathcal{I}d
\end{aligned}
$$

Figure 4: Language of L-systems

Symbol	Meaning
`F`	Draw forward
`B`	Draw backward
`f`	Move forward
`b`	Move backward
`+`	Rotate right
`-`	Rotate left
`[`	Push position
`]`	Pop position

Figure 5: Turtle symbols

3. a drawer which shows the resulting drawing on screen.

The syntax for L-systems is presented in fig. 4. An L-system consists of a header, which describes the various default values along with the axiom, and a set of rules.

L-systems are similar to grammars. The axiom represents the initial state, while the rules associate a symbol to a list of symbols. Unlike traditional grammars, **all** rules are executed at the same time. An example of execution of an L-system is shown in fig. 3.

Finally, some symbols are considered special and have a graphical meaning that pilot a *turtle*. The turtle starts at a given position with a given orientation. Each symbol is interpreted as an instruction to the turtle. For instance, `F` means "draw forward", `+` means "rotate right", and so on. In addition to usual drawing instructions, we have two additional instructions `[` and `]` which to save and recall positions using a stack of positions. `[` pushes the current position to the stack, while `]` pops the last position and teleports the turtle to it. A summary of the symbols with their graphical meaning is shown in fig. 5.

2.1 Implementation/Libraries

The following libraries must be used for this project.

Parsing We strongly recommend to use the small parser combinator library "`ParserCon.hs` provided during the course. It also acceptable to use a richer library, such as `attoparsec`[1].

Graphics The `gloss` library[2] should be used to implement the L-systems drawer. This library uses openGL to easily draw and animate 2D pictures, using a similar API to the SVG library developed in Exercise 2. You can start by using the function `Gloss.display`.

```
Gloss.display :: Gloss.Display -> Gloss.Color -> Gloss.Picture -> IO ()
```

Command line interface Your evaluator should not only accept the L-system to draw, but also a set of parameters to influence its behavior. For instance, *`--generation`* to choose at which generation to stop, *`--animate`* to animate the L-system according to section 3.4, *`--param`* to accept additional parameters for section 3.1, ...

You must implement the command line interface (including documentation!) for your evaluator using the `optparse-applicative` package[3].

[1] https://hackage.haskell.org/package/attoparsec-0.13.2.2/
[2] https://hackage.haskell.org/package/gloss-1.13.0.1
[3] https://hackage.haskell.org/package/optparse-applicative

Tests As all libraries, your L-systems implementation should be tested. We recommend using both handwritten L-systems tests and automatic testing using `quickCheck` (for the evaluator, in particular). A set of L-systems for testing will be provided on the course website.

3 Extensions

Here are a few proposed extensions. You can also invent your own extensions. Further extensions ideas can be found in the ABoP book [Prusinkiewicz and Lindenmayer, 1990].

3.1 Parameterized L-systems

A simple extension to L-systems extends rules with *parameters*, similar to functions. For instance, the command `F` would take a parameters indicating the length of the line. Such parameters can then be used to avoid the need for the default angle and length. You may also consider allowing various computations over the parameters. For example, here is a rewrite of the "shrub" L-system using parameterized L-systems.

```
axiom X
X -> F(1)L[[X]RX]RF(1)[RF(1)X]LX
L -> -(25)
R -> +(25)
F(l) -> F(l*2)
```

Extend your grammar and your evaluator to support such parameterized rules. Make sure to properly check and report errors while evaluating.

3.2 Stochastic L-systems

There are also L-systems where multiple rules are available for a given symbol. The applied rules is decided randomly. For instance, here is a randomized "shrub" L-system which can grow either left or right at each generation. Each rule has a weight that influences the random choice. Such weights could also be combined with the parameters from section 3.1.

```
angle 25
axiom X
X -0.7> F-[[X]+X]+F[+FX]-X
X -0.3> F+[[X]-X]-F[-FX]+X
F -> FF
```

3.3 More Graphic Primitives

`Gloss` supports a wide array of graphic primitives such as colors, text, arcs and arbitrary sprites. Add new symbols to your L-systems language to use these graphical primitives (this might be combined with parameterized L-systems). You can then use these primitives to draw more realistic plants or more artistic tessellations (see Escher's work for inspiration [Wikipedia, 2019a]).

You can also provides alternative geometries than the traditional euclidean plane. It is notably possible to draw L-systems in hyperbolic geometries.

3.4 Animating L-systems

L-systems are often used to simulate plant growth, where each generation represents a unit of time. Introduce animations (using `Gloss.play`) to simulate the plant growth. You may also fix a generation and animate the movement of the turtle. Improve your drawer to support animations and define new L-systems that take advantage of it.

3.5 Static picture export

The `gloss` library uses a picture API that is similar to the one we described in Exercise 2 to create SVG images. Extend your program to support static SVG pictures that can be sent to other people or used in a website. Can you also support animations?

3.6 Advanced automatic testing

One challenging aspect of automatically testing evaluators such as the one developed in this project is to *generate valid programs randomly*. While this problem is a very difficult in general, the L-systems language is sufficiently simple to make this problem accessible.

Create a new `quickCheck` generator for L-systems programs and use it to test your code.

References

Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The algorithmic beauty of plants*. The virtual laboratory. Springer, 1990. ISBN 978-0-387-94676-4. URL http://algorithmicbotany.org/papers/#abop.

Wikipedia. M. c. escher — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=M._C._Escher&oldid=903447150, 2019a. [Online; accessed 29-June-2019].

Wikipedia. L-system — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=L-system&oldid=902901535, 2019b. [Online; accessed 26-June-2019].