DOMAIN SPECIFIC LANGUAGE FOR GENERATING FRACTAL PLANTS WITH STOCHASTIC L-SYSTEMS

Cristian BORIS, Eugeniu POPA, Ana-Maria RUSNAC*, Dumitru MUNTEANU, Liviu MOISEI

Department of Software Engineering and Automatics, group FAF-202 Faculty of Computers, Informatics and Microelectronics, Technical University of Moldova, Chisinau, Moldova

*Corresponding author: Ana-Maria Rusnac, ana-maria.rusnac@isa.utm.md

Abstract: *The main scope of this article is to present the process of designing a domain-specific language, with the purpose of generating fractal plants with the use of Stochastic L-systems. The focus will be on presenting the analysis of the domain, the use of this domain-specific language(DSL) in this domain, describing the computational model, as well as defining data structures, semantic rules, and grammar rules of the language.*

Keywords: *domain-specific language, l-systems, stochastic grammar, grammar, parse tree.*

Introduction

A Domain Specific Language (DSL), in contrast to a General Purpose Language, is a language much more concentrated on a single specialized field, in order to solve a very specific problem in that aforementioned field.

Lidenmayer systems (abbreviated L-systems), first introduced in the year 1968 by biologist Aristid Lidenmayer, were originally created as a theoretical framework for studying the development of simple multicellular organisms, and subsequently applied to investigate higher plants and plant organs [1].

After the incorporation of geometric features, plant models using L-systems became detailed enough to allow the use of computer graphics for realistic visualization of plant structures and development. In the process of designing a domain specific language aimed at the representation of an organic process through the means of algorithmic logic, this article covers three major aspects: domain analysis, grammar definition, and a program example.

Domain-Analysis

L-systems are a recursive, string-rewriting framework, that provide a way of formalizing patterns of development [2]. L-systems are formalized as a tuple with the definition:

$$G = \langle V, \omega, P \rangle \tag{1}$$

Where the V is the alphabet, ω is the starting point, and P is a set of productions.

For this project, L-systems will be used for generating fractal plants. In order to model higher plants, a more complex graphical interpretation of L-systems is needed. The most popular interpretation used today to describe the grammar for an L-system is the turtle interpretation, which can result in realistic modeling of herbaceous plants [1].

One last important aspect this DSL looks to explore is the modeling of plants in such a way that is closest to an actual organic object. This is where stochastic grammar comes in. Simply put, a Stochastic L-system is non-deterministic, it will have probability to its production. If it weren't for these Stochastic systems, the generation of multiple plants would produce a very artificial regularity. A Stochastic L-system, defined by Ec. (2), is an ordered quadruplet, where V, ω, and P, are defined just as before, with the new addition of π, where π is the probability distribution [1]:

$$G = \langle V, \omega, P, \pi \rangle \tag{2}$$

The issue this article tackles is that of portraying realistic organic growth. Modeling the growth of a plant given certain fundamental rules requires particular domain expertise (mostly programming) and is currently done mostly by specialists, but this process is constrained by the availability of experts, the complexity that humans can solve, as well as time. This DSL looks to offer to both experts in the field, as well as regular interested student, an opportunity to quickly observe and study plant growth with minimal understanding, resources, or knowledge of programming of their own.

Language Overview

This particular language has the functionality to take a specific collection of rules, an axiom, written following the rules of l-systems, and based on that generating a fractal plant of stochastic nature in the form of a png file.

The input will consist of an alphabet, the axiom, and a set of production rules, with added probability distribution, since that is the way L-systes are defined as a type of formal grammar. The alphabet is going to be predefined as a set alphabet consisting of "F, X, Y, +, -, [,]", since in terms of L-systems, it is the standard alphabet when constructing fractal plants.

In terms of data structures, it will have 4 main data structures: lsystem, which is a structure that will store the axiom, ruleset, and terminal symbols, the second structure – axiom, defined as the initial string, rule, where the rules are declared by the user, and terminal, which just specifies the termianl symbols and their function. Control structures for the language, the one way the user will be able to control their L-system, is by controlling the number of iterations, thus how big the system is.

Semantic Rules

In order to not get any errors, and for the user to be able to correctly use the language, there are some rules that will be set in place:

- An axiom is the first thing that must be declared.
- A rule/rules must be declared before trying to draw the plant.
- For each rule declare, there must be a probability set in place as well.
- If there are multiple rules for either F, X, or Y the probability for each must be declared in such a way that the sum will be equal to 1.
- A terminal must first specify the terminal itself, and the after the function it performs.
- For both the rules structure and terminals, the components must be separated by commas.
- When generating the plant, the number of iterations must be specified.

Grammar Rules

In general, a programming language grammar, is a set of instructions about how to write statements that are valid for that specific language, there are a few special notations when writing grammar, that help to better distinguish and understand the structure, defined in Tab. 1:

Table 2

Meta Notations

Notation	Description
<foo>	foo is non-terminal
foo	**foo** is terminal
x*	zero or more occurrences of x
x+	one or more occurrences of x
|	separates alternative
?	zero or one occurrence

V_N = {<program>, <LSystem>, <string>, <body>, <axiom>, <ruleSet>, <terminals>, <termianl>, <iterations>, <rule>, <probability>, <angle>, <iterations>, <var>, <alphabet>, <systemAlphabet>, <whole>, <fractional>, <digits>, <action>}

V_T = {A…Z, a…z, +, -, [,], (,), =, 0, 1…9, ., ", begin, end, rule, LSystem.new, axiom, rule, terminal, forwards, push, pop, rotate, iterations }

S = {<program>}

P = {<program> → <LSystem>
<LSystem> → <var> = **LSystem.new**(<string>) **begin** <body> **end**
<var> → <alphabet>$^+$ | <var><digits>
<body> → <axiom><ruleSet><terminals><iterations>
<axiom> → **axiom(“**<systemAlphabet>$^+$**”)**
<string> → “ <alphabet>$^+$“ | “<digits>$^+$” | “<string><digits>$^+$”
<ruleSet> → <rule>$^+$
<rule> → **rule(“**<systemAlphabet>$^+$**”, “**<systemAlphabet>$^+$**”,** <probability>**)**
<probability> → <whole> | <fractional>
<whole> → 1
<fractional> → 0.<digits>$^+$
<terminals> → <terminal>$^+$
<terminal> → **terminal(“<**systemAlphabet>**”,** <action>**)**
<digits> → **0 |1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**
<alphabet> → **a | b | … | z | A | B | … | Z**
<systemAlphabet> → **F | X | Y | + | - | [|]**
<action> → **forward | push | pop | rotate** <angle>
<angle> → <digits>$^+$ | **-** <digits>$^+$
<iterations> → **iterations(**<digits>$^+$**)**
}

Example Program

The input will take on the following form:

```
a = LSystem.new(“My new plant”) begin
        axiom(“+++X”)

        rule(“X”, “F+[[X]-X]-F[-FX]+X”, 1)
        rule(“F”, “FF”, 1)

        terminal(“F”, forward)
        terminal(“[”, push)
        terminal(“]”, pop)
        terminal(“-”, rotate -25)
        terminal(“+”, rotate 25)

        iterations(5)
end
```

If this specific input, generated by the grammar rules stated previously, would be represented through a parse tree, it would take on the form represented in the figure below:

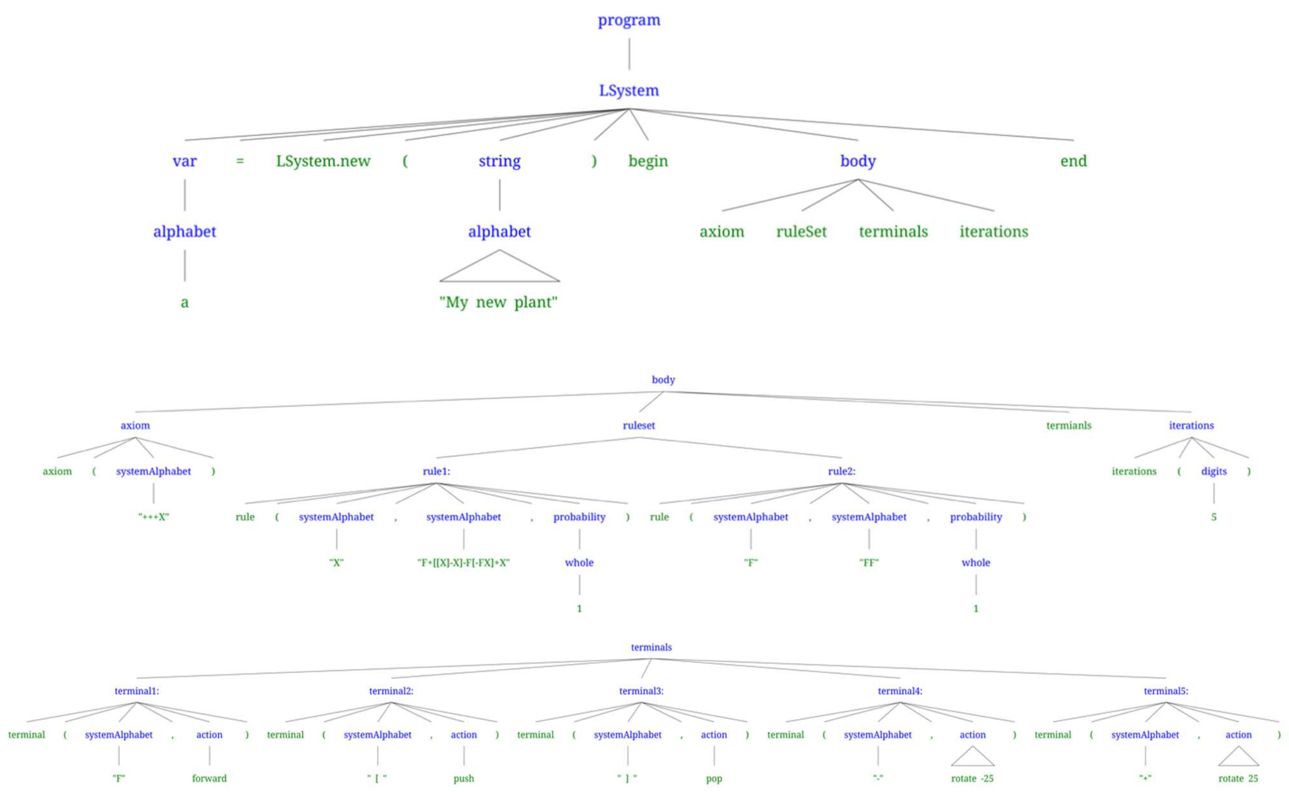

Figure 1. Parse tree

Conclusion

This article discusses the complex topic of designing a domain-specific language, from a simple idea, to a well-developed structure, and it follows the steps necessary to create this language.

This paper focuses on the development of a language that will provide students and experts with a tool to observe and study the development of organic growth, through a DSL that can generate fractal like plants, built with the help of L-systems, and based on stochastic grammar. After this main idea, and a few essential definitions being introduced, the article moves on from the conceptual part to a more practical part, and the most important aspect of designing a DSL, the grammar. The grammar part when designing a language is definitely the one aspect that needs the most precision, knowledge of both programming and the studied field, and understanding of consumers. It is the most vital part, that brings structure and stability to the idea, and make it something tangible.

The process of designing a language is one that requires a deeper understanding of the fields involved, and it does not only provide a look into the inner-workings of what a language needs to be functional, but it also creates essential skills that will be useful for academic success.

References

1. ARISTID LIDENMAYER and PRZEMYSLAW PRUSINKIEWICZ. *The algorithmic beauty of plants.* New York: Springer-Verlag Press, 1990.
2. JORDAN SANTELL. *L-systems,* 2019. [online]. [access 26.02.2022]. Available: https://jsantell.com/l-systems/#stochastic-l-systems.