UNiversidad DIstrital Francisco José de Caldas
Department of Systems Engineering
Faculty of Engineering
Computer Science III

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

CrochetScript

Laura Daniela Muñoz Ipus - Code: 20221020022
Laura Daniela Cubillos Escobar - Code: 20211020045

Abstract

Crochet pattern design is traditionally a manual and time-consuming process, requiring expertise to create precise instructions. To streamline this process, we propose a declarative programming language that generates crochet patterns and exports the results into a structured PDF format. Our approach automates pattern creation, reducing human error and increasing accessibility for both beginners and experts. The implementation of our compiler demonstrates successful pattern generation, ensuring accurate symbol placement and readable instructions while maintaining flexibility for various designs.

Introduction

Crochet is a widely practiced craft that involves creating fabric by interlacing loops of yarn with a hooked needle.

While the craft is accessible, designing crochet patterns requires experience and meticulous planning. Traditionally, crochet patterns are written or traced manually, which can be time-consuming and error-prone. Additionally, variations in notation and terminology in different regions further complicate the process, making it less accessible for beginners.

There are now applications that provide graphical interfaces for pattern design, but they still require manual placement of stitches. In addition, procedural programming approaches have been explored, but they often lack flexibility and require extensive coding knowledge from users. While these tools offer some level of automation, they do not fully encapsulate the declarative nature of pattern definition, where users specify what they want rather than how to construct it.

To address these limitations, we propose a declarative programming language tailored to crochet pattern generation.

Our language allows users to define patterns using high-level constructs that abstract away the complexities of manual design. The compiler processes these declarative instructions and generates structured patterns in a standardized PDF format, ensuring consistency and ease of use. By leveraging parsing techniques such as lexical, syntactical, and semantic analysis, our implementation validates user input and translates it into accurate pattern instructions.

Our approach incorporates computational techniques that are commonly used in programming language design, such as context-free grammars for syntax definition, regular expressions, and abstract syntax trees (ASTs) for intermediate representation. In addition, we use formatting libraries to generate well-structured PDF files that adhere to established crochet notation. This method not only reduces human effort in pattern creation, but also improves accessibility for both beginners and professionals.

In the following sections, we detail the design and implementation of our language, presenting experimental results that show its effectiveness.

Method and materials

In this section, we describe the design of the proposed solution, the technical decisions taken, and the logic behind them. The main goal of this project is to develop a declarative language that generates crochet patterns and exports the results in a PDF file.

The compiler follows a structured approach where user-defined patterns are written using a domain-specific language (DSL). The input consists of a sequence of predefined commands specifying the shape, size, and color of the crochet elements. The output is a visual representation of the pattern, formatted as a PDF document.

To develop this, it is necessary to implement a compiler with lexical, syntactical, and semantic analysis.

0.1. Lexical Analysis

Lexical analysis is the first stage of compilation, where the input string is divided into **lexemes and tokens**. Each lexeme is categorized into a token class based on its function. The following table outlines the lexemes and their corresponding tokens in the language:

LEXEMES	TOKENS	MEANING
START	keyword	Starts the pattern description – starts the code
addCircle	Figure	Add a circle to crochet pattern
addSquare	Figure	Add a square to crochet pattern
addTriangle	Figure	Add a triangle to crochet pattern
addHeart	Figure	Add a heart to crochet pattern
RED	Color	Option for figure color
PINK	Color	Option for figure color
BLUE	Color	Option for figure color
BLACK	Color	Option for figure color
	Size	Option for figure size
(	Special char	Opening parenthesis
)	Special char	Closing parenthesis
END	keyword	Ends the pattern description – ends the code

Cuadro 1: Lexemes and Tokens used in the language

0.2. syntactical analysis

What is developed when doing the syntactic analyzer is taking the tokens from the lexical analysis and checking that the structure of the code is valid according to the grammar of the language. It is represented by generative grammars and by means of syntactic trees (AST - Abstract Syntax Tree). This phase is very important when making a compiler because it detects syntax errors, such as an if without parentheses or a missing semicolon.

The generative grammar of the developed language is presented below:

Generative Grammar

```
<S>  -> "START" <FIGURE_SEQUENCE> "END"

<FIGURE_SEQUENCE>  -> <FIGURE> "(" <SIZE> <COLOR> ")" <FIGURE_SEQUENCE>
                    | <FIGURE> "(" <SIZE> <COLOR> ")"

<FIGURE>  -> "addCircle" | "addSquare" | "addTriangle" | "addHeart"
```

```
<SIZE>  -> "1" | "2" | "3" | "4" | "5"

<COLOR>  -> "RED" | "PINK" | "BLUE" | "BLACK"
```

This grammar ensures that every pattern begins with the keyword `START`, followed by a sequence of figures with their respective size and color, and ends with the keyword `END`.

To verify that the parser part is correct and in turn the generative grammar, decision trees of different sentences were made, which showed a correct validation, as the following example:

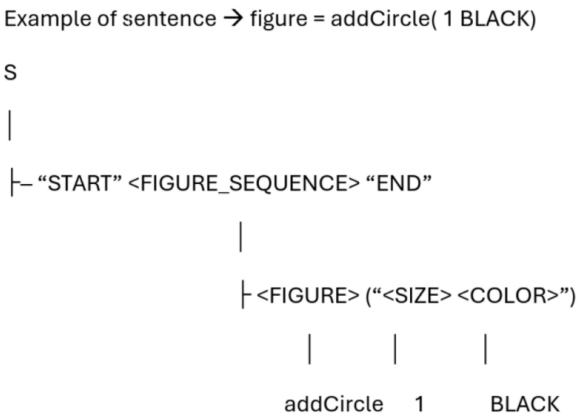

Figura 1: Derivation tree of a sentence

0.3. semantic analysis

In this phase of the compiler, the semantic analyzer seeks to ensure that there is consistency in the syntax and to verify that the operations make logical sense within the program. To achieve this, functions are created in the creation of the language that allow the detection of semantic errors, such as incorrect operations or incorrect assignments.

0.4. Implementation and Tools

The compiler was implemented in Python using two key libraries for the phases of the compilation process. The re library (regular expressions) helped in the lexical analysis, allowing to detect patterns such as keywords, variables and operators. Reportlab allowed the export of the generated crochet patterns in a PDF format, presenting the results in a graphical and understandable way for the user.

Results

To validate the correctness and reliability of our compiler, we performed a number of tests, including unit tests and integration tests. Unit tests focused on verifying the functionality of lexical, syntactic, and semantic analysis, ensuring that the compiler correctly tokenizes input, validates grammatical structures, and detects semantic inconsistencies.

Integration tests ensured that the system components (parsing, abstract syntax tree (AST) generation, and PDF export) worked together seamlessly. These tests confirmed that correct input patterns consistently generated structured and readable PDF outputs.

Conclusion

The development of CrochetScript successfully demonstrates the feasibility of using a declarative programming language for crochet pattern generation. By automating the pattern creation process, our compiler significantly reduces human effort, minimizes errors, and improves accessibility for designers of all skill levels. The implementation of lexical, syntactic, and semantic analysis ensures that user-defined patterns are both valid and accurately translated into structured PDF instructions.

Testing results confirm the robustness of our approach, with high success rates in unit and integration tests, as well as positive feedback from real users. Compared to existing manual and graphical tools, our solution offers a more structured, error-free, and time-efficient alternative for crochet pattern design. Future work could focus on expanding the language's expressiveness, optimizing pattern visualization, and integrating support for interactive design tools. The results validate our hypothesis that a domain-specific language tailored for crochet patterns can enhance creativity while simplifying the technical aspects of design.

Bibliography

@miscCSIIIRepo, author = EngAndres, title = Repositorio de Computer Science III, year = 2025, howpublished = https://github.com/EngAndres/ud-public/tree/main/courses/computer-science-iii,