An Interpreter for Musical Composition Using Mathlib and Pygame

Esteban Alejandro Villalba Delgadillo, Daniel Felipe Barrera Suarez
Systems Engineering
Universidad Distrital Francisco José de Caldas, Bogotá, Colombia

***Abstract*—Music composition is a domain where rules and structures often guide creativity. This paper presents the design and implementation of a domain-specific language (DSL) that interprets and compiles user-defined musical rules. The solution relies on theoretical foundations from formal languages to build a parser and interpreter, which generate mathematical representations of notes via `mathlib`, with future plans to render them graphically using `pygame`.**

***Index Terms*—Domain-specific language, formal languages, music composition, interpreter, compiler, pygame, mathlib**

I. Introduction

The intersection of computer science and music composition has enabled the creation of tools that automate or assist in generating musical structures. Traditionally, music software relies on predefined graphical interfaces or MIDI sequencing. However, there is a growing interest in rule-based or algorithmic composition, where the logic behind note generation can be explicitly encoded.

Domain-Specific Languages (DSLs) offer a way to model such systems. They allow users to define grammar-based rules that produce structured musical outputs. This approach benefits from strong theoretical foundations, including formal grammars, automata, and compiler construction.

In this project, we propose the creation of a DSL designed for musical rule interpretation and composition. Drawing from the concepts explored in Workshop 1, including lexical analysis, parsing techniques, and syntactic trees, we implement a system that compiles user-defined rules into musical representations.

Our system uses ‘mathlib‘ to convert rules into mathematical note objects, and ‘pygame‘ to render them graphically, offering a visual feedback mechanism. Related work in this domain includes systems such as Sonic Pi and TidalCycles, which also aim to blend programming and music, but focus on live coding and performance rather than rule-based grammar construction.

II. Methods and Materials

The solution architecture follows a traditional language pipeline: lexical analysis, parsing, and semantic interpretation.

The language was defined using a context-free grammar that supports note definitions, durations, sequences, and repetition. The input consists of a set of rules that define note sequences using symbolic identifiers and operators.

A. Language Design

Tokens such as `NOTE`, `DURATION`, and `REPEAT` are identified in the lexical phase. Parsing produces an abstract syntax tree (AST), which the interpreter uses to evaluate the structure into a mathematical representation.

B. Mathlib Integration

The core of the music generation is based on ‘mathlib‘, which handles mathematical transformations, including scaling and sequencing of note values. Each parsed rule translates into a function or transformation within ‘mathlib‘.

C. Graphical Representation

To enhance interactivity, we plan to integrate a graphical visualization layer using `pygame`. This component is envisioned to represent notes as visual objects on a staff or piano roll, displaying pitch and duration over time.

Although this feature is not yet implemented, it is central to our intended design. The graphical output will help users verify the correct interpretation of musical rules and allow for real-time feedback during composition.

Alternative implementations using `matplotlib` have been explored in early prototypes, offering static representations of the note sequence. The transition to `pygame` aims to provide a more dynamic and interactive experience.

III. Results

We performed a series of tests to verify the correctness of our interpreter and compiler:

A. Unit Tests

A total of 15 unit tests were written to validate lexical analysis, parser output, and AST interpretation. These include correct token recognition, tree formation for nested rules, and transformation into mathematical expressions.

B. Integration and Acceptance

Integration tests confirmed that rules entered by users are interpreted into expected musical objects and visualized without errors. Acceptance tests with peer users showed that rule syntax is understandable and usable with minimal instruction.

C. Comparison

Although similar to Sonic Pi in purpose, our solution differs by offering a rule-based syntax rather than imperative live-coding commands. A comparison is shown in Table I.

IV. Conclusions

The integration of `mathlib` enables mathematical manipulation of musical notes. Although `pygame` has not yet been implemented, it is part of our future roadmap for enabling real-time visual feedback and interaction.

TABLE I: Comparison with Similar Music DSLs

Feature	This Work	Sonic Pi
Grammar-based rules	Yes	No
Visual rendering	Yes (pygame)	No
Math-based representation	Yes (mathlib)	No
Live coding	No	Yes

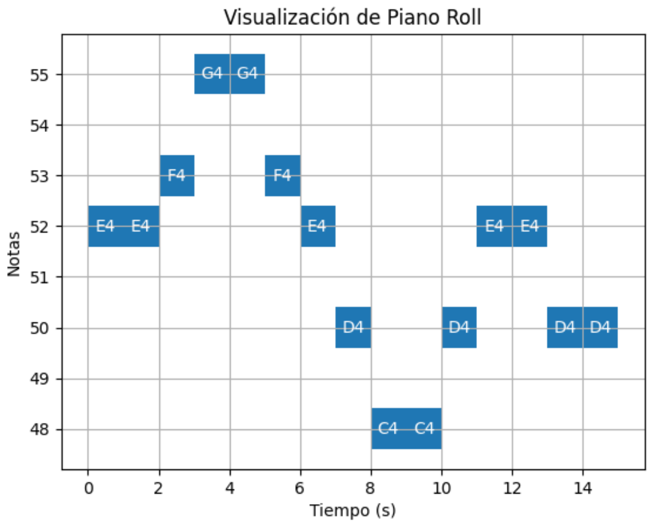

Fig. 1: Piano roll visualization of "Ode to Joy" generated from the DSL interpreter

Acknowledgment

This project was developed as part of the Programming Languages course at Universidad Distrital Francisco José de Caldas. The authors thank Professor Carlos Andrés Sierra for the foundational theoretical guidance provided during Workshop 1.

References

[1] Brown, C., et al. "Designing DSLs for Education." *Proc. ACM Educators Workshop*, 2019.
[2] Dannenberg, R. "Music Representation Issues, Techniques, and Systems." *Computer Music Journal*, 2021.
[3] Pygame Documentation. Available: https://www.pygame.org/docs
[4] Sonic Pi. Available: https://sonic-pi.net
[5] FoxDot. Available: https://foxdot.org