Proposal: ABCD - ABC Music Notation extendeD with programming language features*

Trent Jeffery
The University of British Columbia
Vancouver, B.C.
y3f0b@ugrad.cs.ubc.ca

Ziyang Jin
The University of British Columbia
Vancouver, B.C.
f4a0b@ugrad.cs.ubc.ca

Victoria Wang
The University of British Columbia
Vancouver, B.C.
y7x9a@ugrad.cs.ubc.ca

Yifan Yang
The University of British Columbia
Vancouver, B.C.
p7w9a@ugrad.cs.ubc.ca

ABSTRACT

This is our project proposal for our new Domain Specific Language - ABCD, the extended language that adds programming features to the classic ABC notation.

KEYWORDS

Programming Language, Music, ABC Music Notation

ACM Reference Format:
Trent Jeffery, Ziyang Jin, Victoria Wang, and Yifan Yang. 2017. Proposal: ABCD - ABC Music Notation extendeD with programming language features. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Music is one of the most beautiful and ancient languages humans have created. With the advancement of technology, many solutions have been designed to facilitate the creation and spread process of music. ABC is one of the most widely used music notation forms. Even though it has great features to help musicians generate formal music sheets from scratch, its nature of being a markup language has limited its capacity for effective modification.

Our group plans to design a domain specific language called ABCD that extends ABC with variables, functions, conditionals and loops. Our goal is to allow users to modify an existing ABC file or create a new piece by writing scripts in ABCD that complies to ABC. This proposal will elaborate on how it will fulfill the project requirement and what steps we would take to achieve the goal.

2 HOW IT FULFILLS THE PROJECT TYPE REQUIREMENTS

The purpose of our project fits into the fifth project type. We will design a DSL (Domain Specific Language) called ABCD which empowers users to edit and modify music sheets written in ABC music notation more effectively and efficiently. The users of ABCD will be professionals who use ABC to write their music. ABCD could help users in many ways, and some general user cases are listed below:

- A trumpet player may want a program that transposes songs for C instruments to scores for B flat instruments
- A composer writing a score in ABC may wish to change the key of the last x bars of their song
- An arranger may wish to add a harmony to every instance of a certain note
- A producer may wish to generate part of a song or a whole song programmatically

All of these can be easily accomplished as programs written in ABCD. ABC alone is not powerful enough to do these programmatically – it would have to be done manually. An alternative solution would be using another programming language, but extra efforts on parsing and compiling ABC would be required, whereas ABCD would support those natively.

3 HOW IT COULD BE A FULL 100% PROJECT

We intend to design a Domain-Specific Language extending the ABC music notation language. ABC is similar in power and purpose to LaTeX: it allows a user to hand-type a document that represents a piece of sheet music, and can be compiled to a PDF or played aloud using appropriate software.

Our language, ABCD, will enhance ABC with variables, functions, conditionals, and loops, empowering users to programmatically modify existing ABC documents and generate or author new music more easily. Variables will allow snippets of music code to be built up, modified, and used multiple times. Functions will provide a way to transform or generate music code in a way thatâĂŹs reusable over different pieces of music code. Conditionals and loops provide control flow mechanisms that make it much easier to apply changes selectively or repeatedly.

ABCD will be used to write scripts that compile into ABC files after modification or creation. By compiling to ABC files, our users can leverage existing software that works with ABC, including music players and pdf compilers. ABCD should have native support for importing ABC files, allowing users to easily modify existing ABC scores. To support the use of "library files" containing useful

*The authors' names are ordered by last name.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
Conference'17, July 2017, Washington, DC, USA
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

definitions, the language should also support importing other ABCD files.

An ABCD file may contain definitions of variables and functions as well as ABC notation. When compiled, functions will modify, add, or remove ABC code in the script, and the script will evaluate to one or more ABC documents.

A full implementation of the language would involve:

(1) An EBNF defining concrete syntax for:
- Variables
- Functions
- Loops
- Conditionals
- Import statements
- ABCD data types (especially types that arenâĂŹt ABC)

Any other syntax constructs required to support these features
(2) An ABCD parser
(3) An abstract syntax implementation
(4) A compiler to produce ABC files from the ABCD abstract syntax

We would likely write our parser and compiler in python to harness its string processing capabilities, as ABC has a plain-text format where characters often need to be considered individually. For a final project, we would likely also write a small ABCD program to demonstrate a possible use case for ABCD and showcase some of its capabilities. This would probably be a program that applies some transposition, addition of harmonies to certain notes, and some rhythm changes, and we would demonstrate by applying the program to a small ABC tune and playing the result or printing it as a pdf.

4 APPROACH TO GET TO 80% MILESTONE

The 80% milestone is a background research report. Based on the deadline (Nov 10, 2017), we plan to divide our process into 3 phases. We will communicate our daily progress online. Our planned project meetings are on Monday in class and tutorial, and 5 - 7 p.m. and Wednesday 5 - 6 p.m., and there may be more meetings during the week based on our progress.

4.1 Phase 1: Get familiar with ABC notation

Time Oct 27, 2017 — Oct 31, 2017
Familiarize ourselves with ABC notations features. Read through the abc notation website. Download the software related to ABC and play with it, for example, easyABC. Each team member will write 2 small music pieces(can be very short) using the existing ABC notation, submit them to the git repo, and demo them on the project meeting on Oct 30. Discuss our own music-writing experience and technical difficulties with ABC and document our conclusions in a google doc.

4.2 Phase 2: Read the full ABC language specification

Time Nov 1, 2017 — Nov 5, 2017
Read through the full ABC language specification, record our thoughts and discoveries in a google doc. Discuss online or in person to decide which parts can be used directly or after some modification in our ABCD language, and document these in our draft of ABCD specification. Record in google docs why such choices have been made. Estimate what we need to do for ABCD. The result of this phase will be a pdf file that is part of our ABCD language specification, which lists all the borrowed notations from ABC.

4.3 Phase 3: Criticize ABC and develop a draft ABCD

Time Nov 6, 2017 — Nov 9, 2017
Read discussions on stackexchange and blogs to see people's opinion and suggestion on ABC. List ABC's advantages and disadvantages and make clear the value of developing ABCD. Integrate our conclusions in phase 1 and phase 2. Discussion our potential project. Go through the citations. The result of this phase will be our background research report.

5 LIST OF STARTING POINT DOCUMENTS

The following is a list of sources we would consult during the project.

First of all, http://abcnotation.com is a general site for all ABC related documents. In particular, http://abcnotation.com/examples contains straightforward specifications (illustrated by examples) for how ABC works as a markup language that represents sheet music. http://abcnotation.com/software provides a list of command lines and software packages that can read or edit ABC.

Another source that allows us to understand ABC better and, more importantly, implement ABC software applications is the ABC wiki page: http://abcnotation.com/wiki/abc:standard:v2.2 In addition to the formal specifications of ABC, this site also provides information about ABC extensions, file readers, transposition operations, transposition macros, etc. It allows us to see the various features already supported by ABC, such as transposition. We would need to think about what other features we can implement and/or how to improve the existing features in our language.

http://www.mandolintab.net/abcconverter.php is an online converter that we can use for converting ABC into sheet music and export the file as MIDI, PDF, etc. It could be useful for testing whether a ABC file is valid. We may also use EasyABC (http://www.nilsliberg.se/ksp/easyabc/), an open source ABC editor, if we find the converter not sufficient.

We may also learn about the limitations of ABC at https://music.stackexchange.com/questions/23841, where many ABC users and software developers share their experiences of working with ABC, such as in parsing and in practical use.