Music Generation using Parametric Grammar

Rajvi Mehta
Computer Department
Sardar Patel Institute of Technology
Mumbai, India

Aayushi Acharya
Computer Department
Sardar Patel Institute of Technology
Mumbai, India

Pratik Bonde
Computer Department
Sardar Patel Institute of Technology
Mumbai , India

Mr. Anand Godbole
Computer Department
Sardar Patel Institute of Technology
Mumbai , India

***Abstract*—Music plays a vital role in the life of human. Different components of Music are elements, pitch (which governs melody and harmony), rhythm (and its associated concepts tempo, meter, and articulation), dynamics, and the sonic qualities of timbre and texture. Music composition demonstrates a form of creativity. In this paper an attempt is made to automate the music generation process using controlled grammar and results were tested using an Android based application L-Musica.**

Keywords—pitch , rhythm , tempo , meter articulation timbre , texture , grammar.

I. INTRODUCTION

In automata theory, a language is defined as the set of strings generated by a grammar. The grammar consists of start symbol, production rules, a set of symbols called as terminals and non terminals. Language derived from symbols may be accepted or rejected. Decision of acceptance and rejection is based on whether the string is derivable from grammar or not.

Melody also consists of basic symbols called tones; namely Sa, Re, Ga, Ma, Pa, Dha, Ni. Whenever the music is generated these notes are played in some order with different pitch, tempo and texture.

To generate notes to be played one after the other, one can make effective use of grammar. After the string derivation the symbols (terminals and non terminals) in derived string can be mapped to symbols in music which can be played later.

For example consider the grammar shown below.

A -->aBc
B --> CD
C --> a
D --> d

Then derived string can be aBc which can be expanded to aCDc. String aCDc can further expanded to aadc. If symbol ‘a’ is associated with Sa, ‘d’ with Re and ‘c’ with Ga then notes that will get played will be Sa , Sa, Ga , Re.

II. PARAMETRIC GRAMMAR

A. Parametric Grammar

In the above example if used as it is, it can derive the string to be played in an uncontrolled fashion and which will be always static in nature, once the production rules are fixed. To have additional control on the derived string following parameters can be added.

1. Use of control characters

2. Association of Iteration with tempo by which a note can be played.

B. Use of Control Characters

Control characters will be characters which will be made part of production rules. An appropriate meaning or action/behavior can be associated with it. This in turn, can control the derivation of string at current iteration. For example character ‘^’ can be used to stop the expansion of all characters (non terminal as well as terminals) appeared before it. Control character ‘+’ can be associated with a different action like, the symbol before ‘+’ can be expanded at the next iteration.

C. Association of iteration number with tempo of a note

The process of deriving the string goes through several iterations. During the derivation, once the symbol gets derived, its iteration number can be maintained internally. Once the final string to be played is derived, the tempo of each note or the duration for which note can be played can be varied.

III. L-MUSICA

A. LMusica an Android application was developed to test the results

The application is based on the research work on how to generate music and implement it using Android OS. The music player, which uses the front and back end architecture, is divided into the part of music playback and the part of the player interface. The music is played on the basis of the grammar where the characters correspond to different musical notes. The application gives user two choices, either to build his own music or to make use of the set of inbuilt

music. The first page gives a list of different tracks stored in the module.

With a single click, another activity opens which shows the description of the grammar of that particular track along with the freedom of entering the number of iterations the user wants to run that grammar for. The user can also experiment with generating his own music by giving his own grammar as the input and choosing the algorithm. Different approaches were developed in order to generate the music. The Fig. 1.shows the screenshots of the application built.

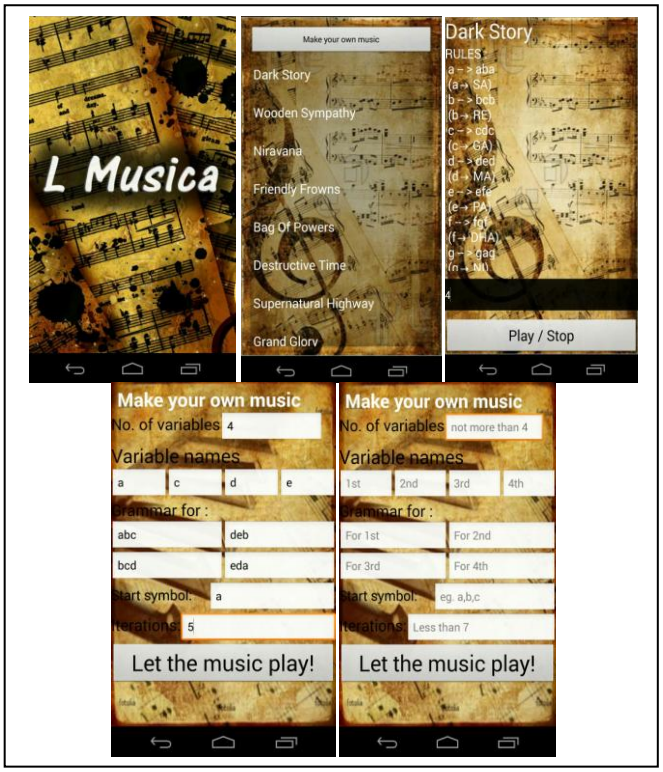

Fig. 1. Screenshots of the application

B. *Architecture of LMusica*

L-Musica consists of 3 modules as shown in Fig. 2.

Input module: This module provides a user interface with the help of which the user can feed the different production rules. Each production rule has L.H.S. and R.H.S. and it also accepts the number of iterations to be carried over to derive the string to be played.

Production Modifier: This module introduces the control character in the production rules entered by the user and inserts them at predetermined positions. These altered production rules are fed to the next module called Generator.

Generator : Generator module generates the string of symbols to be played by choosing one of the generating functions as per the users wish. Depending on the selection of the generating function it generates the string to be played and replaces them with actual notes by predetermined association.

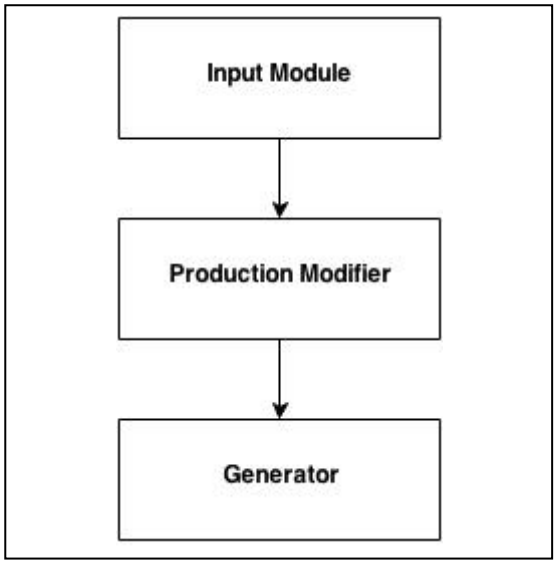

Fig. 2. Diagram of architecture

IV. DIFFERENT APPROACHES USED FOR MUSIC GENERATION

A. *Different approaches were tried for generating the derived string*

*1) Approach 1:*As mentioned in section I, the music generation algorithm uses the concept of grammar and uses set of production rules. All the production rules are outlined in such a way that the right hand of each production rule is having odd number of characters. While generating the derived string, the middle character of string whose derivation is in progress is replaced by its production rule and the other characters in the string will be kept as it is. By using this constraint, it is possible to generate the 'Alankar' of the classical music.

It has been also observed that if all the production rules exhibit a palindrome pattern, the derived string will also be palindrome in nature.

Algorithm uses special characters like '^' (caret) and '+' (plus). The interpretation for the symbols is done as follows: The symbol '^' (caret) when appears in right hand side of the production rule, the string before it will be kept intact.

The symbol '+' when appears in right hand side of production rule, the character just before '+'(plus) will be replaced by its corresponding production rule and other symbols will remain unchanged. If character just before '+' derives an empty string then the character before it will be considered for further expansion. This process of moving in the left direction will continue till we find the character which derives a non empty string.

While deriving the string, the algorithm will also keep track of the levels at which the character is expanded. While playing the note, the tempo can be varied. Refer to example shown in Fig. 3.

*2) Approach 2 :*In second approach there are 2 components used.

a) Set of production rules

b) Reference string

Reference string is formed using the left hand side of each production rule, and will follow the same sequence in which production rules are specified. It is a sequence of all the left hand side of the production rules.

The string derivation process will choose the last character of the string. Algorithm will append to the last character, the string which is RHS of the production rule of the symbol, which appears just before it, in the reference string.

It is possible that this process may select a such a production rule which may or may not derive the empty string. If it is deriving the non empty string, then RHS will be appended to the current string. If it is deriving empty string then, the next character in left direction in reference string will be selected for replacement. This process will be continued till the character which produces non empty string is found. If we reach to extreme left of reference string, then the string derived so far will be abandoned and the new derivation will continue from RHS of production rule, which appears just before it but in reverse. The reverse process goes on till the time a non-empty derivation is found. Once an empty derivation is encountered and there are no further symbols in the reference string to the left, the string is abandoned and the algorithm will start from the beginning.

In the algorithm, special characters used are '^'(caret) and '-'(minus). Here, the symbol caret would indicate that the string before it, would not be replaced or removed and it will be kept as it is. The '-' would mean that only the particular character before it will be looked up into the sequence and the character before it in the sequence would be used. Its corresponding production rule will be appended to the current expansion. Refer to example shown in Fig. 4.

*3) Approach 3 :*In this, the algorithm is divided into two stages.

a. Appending Stage

b. Discarding Stage

The two stages occur alternatively and are independent.

The algorithm starts with the appending stage. The string derivation process will append the production rule of the last symbol in the string. If a symbol derives empty string, there is a switch to the discarding stage. In the discarding stage, in one iteration, the last character in the string is removed. If the string becomes empty after few iterations, the process switches to the appending stage. The first symbol in the appending stage will always be the start symbol.

In this algorithm, special symbols used are '&'(and), '^'(caret), '$'(dollar). The symbol '&' means that the character before the '&' is taken and the production rule for that symbol is appended to the end of the string. The caret '^' would mean the same, as defined in previous approach. The dollar '$' would replace the last character of the sequence with a '$'. When there is only a '$' left, it is replaced by the start symbol.

Refer to example shown in Fig. 5.

B. *Implementation Details*

The process starts with the start symbol and there is a limit to the number of iterations.

For a particular iteration, when the string is being processed, the generator will look for the special symbol from left to right. Once the symbol is encountered, the string before it will be kept in a buffer and even the symbol will be added in the buffer. Now the counter will look for the end of the string or the next symbol. If either of it is encountered, the string in the buffer is processed according to the special symbol. This process is repeated till the end of string is reached. In the end, we obtain a complete string.

Each letter in the string represents a 'swara'(in Sanskrit) or a tone of the classical music. Hence when the complete string is processed, each letter is substituted by its tone and the music is played accordingly. Only the letters denote a tone and not the special symbols.

Each approach represents an 'Alankar' of the classical music.

For the next iteration, the same process is repeated. A constraint is set on the number of iterations. When the counter reaches that value, the symbols will not be expanded anymore.

If the inbuilt music is played, the corresponding production rules have the special symbols within it. If the user enters the grammar, first it will be sent to a modifier unit, and later to the generator. It is passed to the generator module in both the cases: inbuilt or user-made, which will interpret the symbols and carry out the necessary functions.

C. *Specific Examples*

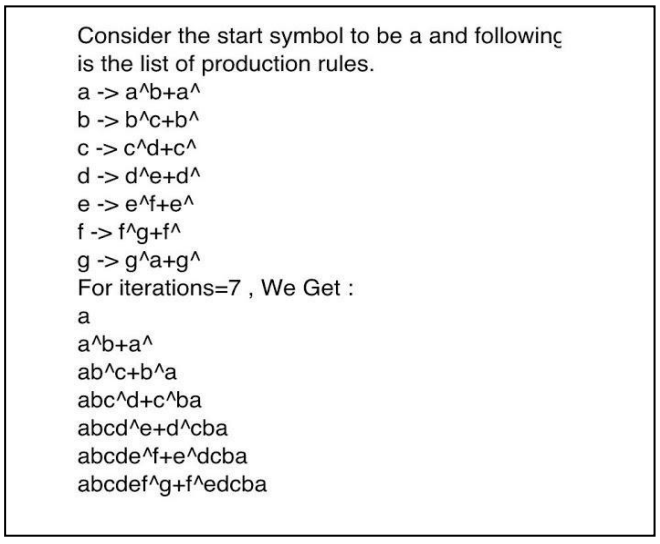

Fig. 3. Example for approach 1

*1) Explanationof example for approach 1:*The process starts with the start symbol, and in this case it is 'a'. For instance, consider the current string under expansion, 'a^b+a^'. After the processing of '^'(caret), the string would be 'ab+a^'. The special character '+' will be processed next, before '+' , character b is present, so by the definition of '+', that character b will be replaced by its production rule. Currently the string will be 'ab^c+b^a^'. The last special character is caret after a. The final processed string will be 'ab^c+b^a'. As each character except the special characters refers to a musical tone, the music will be played.

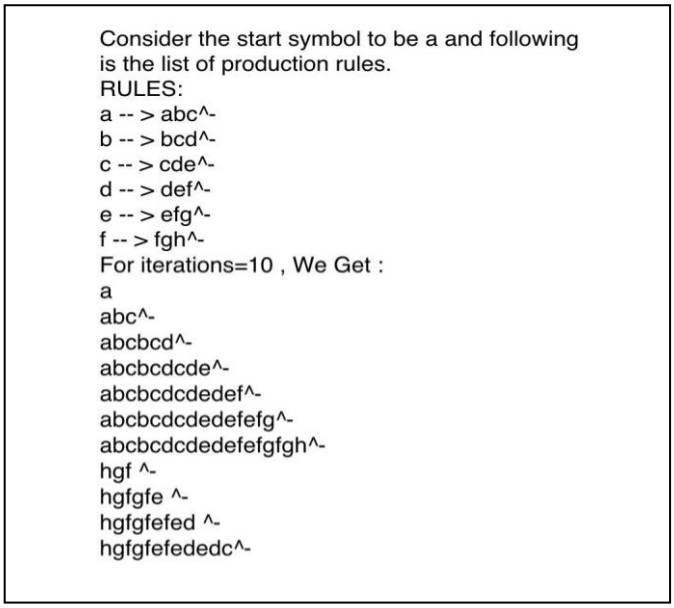

Fig. 4. Example for approach 2

2) Explanationof example for approach 2: The sequence for this example will be 'abcdefgh'.

The start symbol is a, so the process of derivation starts with a. For instance, consider the current string under expansion is abc^- then all the characters before carat will remain intact. After carat is processed, the string is abc-. The '-' will be interpreted as follows: the character before the minus symbol is c; the symbol before c in the sequence (abcdefgh) is b. Hence the production rule of b will be appended to the pattern, which is bcd^-. The final string will be abcbcd^- .

When the current string under expansion is, 'abcbcdcdedefefgfgh-', the character before - is h and the character before h in the sequence is g. There is no production rule for 'g'. Here, the last available production will be checked, which in this case is f → fgh and the pattern will be generated from the next step by reversing the RHS of this production rule excluding the special symbols. Thus, in this case, it will become hgf^-(or hgf-). For the next iteration the string under process would be 'hgf^-'. The character before '-' is f and the symbol before f in the production rules is e → efg. The reverse of the production rule of e should be appended(excluding the special symbols). Hence, the final string would be 'hgfgfe^-'.

3) Explanation of example for approach 3: The start symbol is a, so the process of derivation starts with a. The stage of appending is currently in progress. Consider a string, say 'abcdefg^&' to be processed. The characters before caret '^' will be kept intact. After caret '^' is processed, the intermediate string would be 'abcdefg&' .In order to process '&' the character before the '&' is taken and the production rule for that symbol, in this case, 'g' is appended to the end of the string. The final string would appear like 'abcdefgh$'.

The string under process will now be 'abcdefgh$', the discarding stage will start now. This happens because 'h' derives an empty string. Also the definition of '$' supports that. The control character '$' is processed. The last character in the string, which is 'h' is replaced by '$'. The discarding stage continues until only '$' is left in the processing string.

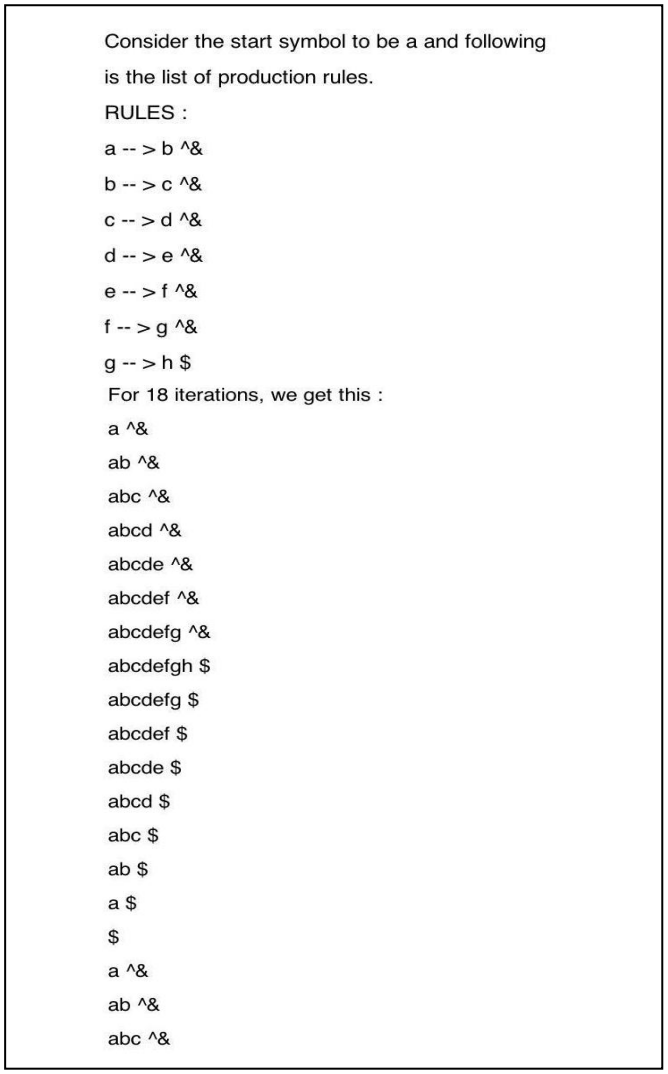

Fig. 5. Example for approach 3

V. CONCLUSION

Music was created by using parametric grammar and various approaches can be taken as per the need of the user. different approaches were designed in order to play the music. Set of control characters were introduced which helped in the implementation of the approaches. Some control characters were specific to a particular algorithm whereas some were interdisciplinary. Each music that was generated corresponded to an 'Alankar' in the classical music. The Android application was successfully made. LMusica was shown to run in the emulator as well as on the actual mobile device running on Android Operating System. All the approaches were tested using the application.

Set of control characters with specific meanings and uses were introduced in the grammar rules which helped in the implementation of the approaches. Some control characters were specific to a particular algorithm whereas some were interdisciplinary. Each track of music that was generated corresponded to an 'Alankar' in the classical music.

The Android application, LMusica was made in which the available tracks as well as the user-defined tracks can be successfully played. LMusica was shown to run in the emulator as well as on the actual mobile device running on

Android Operating System. All the approaches were tested using this application. Various tools such as SDK and ADT plugin were studied and implemented successfully. The module where user defines the different grammar rules and iterations for playing a type of music was also tested with varied combinations of grammar provided that the constraints mentioned for that particular algorithm were fulfilled.

VI. FUTURE SCOPE

With respect to LMusica, we can expand the application by adding more grammars which will give the user additional options for the tracks to be played from. Additionally, the notes of the sounds which are assigned to the symbols of the grammar can have higher and lower notes for the same sound (in the same level) which will give added variations to the tracks. Furthermore, there is a possibility of adding notes of other musical instruments. This can be done in two ways: all the notes in a particular grammar are of the same instrument or different symbols of the same grammar can be attributed to different musical instruments.

If we are able to design grammar which will generate the 'Pakad' of a Raga then Raga can be identified very easily. This could be useful in the classical music domain where the already knowledgeable users can take the help of the application to identify certain ragas to conform and the novices can use it to identify ragas conveniently and learn from them.

Moreover, the ability to construct a musical theory from examples is a great challenge, both intellectually, and as a practical means for a range of new and exciting applications. Machine learning is the process of deriving a model, such as a set of (stochastic) rules, from data samples. Hence, extant music pieces could be used to create new pieces through machine learning and statistical analysis methods. Through analysis and prediction, application can generate certain acceptable grammar for pleasant music. In addition to this, in the Make your Own Music module, the user could be given suggestions for certain grammar rules on the basis of the past history or what might suit well in combination with the already fed in grammar rules.

REFERENCES

[1] Prusinkiewicz Przemyslaw and Hammel Mark. (1994). “Language-Restricted Iterated Function Systems, Koch Constructions, and L-systems.” In New Directions for Fractal Modeling in Computer Graphics, SIGGRAPH ’94 Course Notes. ACM Press, 1994.

[2] J. Mishra (2008). Classification of Linear Fractals through L-System, First International Conference on Emerging Trends in Engineering and Technology, vol. 1, no. 5, pp. 16-18.

[3] P. Meyer (1993). The Fractal Dimension of Music. Senior Thesis, Columbia University.

[4] Prusinkiewicz Przemyslaw (organizer). (2003). “L-systems and Beyond.” SIGGRAPH 2003 Course Notes .

[5] Roads, Curti. (1979). “Grammars as representations for music.“Computer Music Journal 1979, Vol. 3 No 1.

[6] Rowe, Robert. (1993). “Interactive Music Systems: machine learning and composing”, Cambridge, MA: MIT Press, 1993.

[7] Google books - L-System Fractals

[8] http://algorithmicbotany.org/papers/abop/abop-ch1.pdf

[9] http://algorithmicbotany.org/papers/score.icmc86.pdf

[10] http://www.sonology.org/NL/thesis-pdf/Stelios%20Manousakis-Musical%20L-systems.pdf

[11] http://www.iaeng.org/publication/WCECS2013/WCECS2013_pp808-812.pdf

[12] http://www-users.cs.york.ac.uk/susan/bib/ss/nonstd/eurogp05.pdf

[13] http://www.music.columbia.edu/~luke/dissertation/dissertation.pdf

[14] www.dipanjandas.com/files/frsm_gen.pdf