2018 IEEE 18th International Conference on Advanced Learning Technologies

Learning Recursion from Music and Music from Recursion

Nikhil Sharma
Department of Computer Science and Engineering
Indian Institute of Technology Tirupati
Tirupati, India.
Email: tee15b014@iittp.ac.in

Sridhar Chimalakonda
Department of Computer Science and Engineering
Indian Institute of Technology Tirupati
Tirupati, India.
Email: ch@iittp.ac.in

***Abstract*—Recursion is a fundamental concept in several courses in computer science but is often one of the most confusing concepts for beginners. On the other hand, music is universal and appeals to most people. This paper is a novel attempt to understand and utilize the synergy between music and computer science to teach recursion to freshman students. The paper presents the retrograde strategy and collaboration techniques used during composition of music and explains how these ideas can be used to learn recursion and stack data structure. We taught recursion to 30 first year undergraduate students in computer science using the normal approach and then demonstrated the control flow using the retrograde strategy. We found that around 81.2% preferred correlation with music. We see this as an initial step towards a deeper correlation of composing music and composing code.**

I. Introduction

The correlation between music composition and code composition is not new [1]. Coding has been linked with many other domains as well. A synergy between two domains often makes it easier to understand the other if we have knowledge of one [1]. Overmars et al. and Mariana et al. present an approach to teach computer science using game design and a digital game in their papers [2], [3] respectively. Other studies that bring out synergy between computer science and other domains include study between agents and data mining [4], web service technologies and simulation [5], and between medical informatics and bio-informatics [6]. The retrograde strategy that we will be talking about in this paper has been applied in the medical domain. We present in this paper one such synergy between computer science and music.

Programmers and Musicians believe that their compositions are esoteric. The advance and diverse nature of each of the fields makes their synergy a bit involved, yet extensively studied by the researchers [7], [8]. Every algorithm or a data structure is in itself an expression of a problem solving strategy of the designer. It takes a similarly creative and different approach to compose a melody that never existed before. The algorithms are rhapsodies for the programmers whereas the musicians have their own algorithms to create a melody. We describe one such strategy followed by the musicians which is also used in several places in algorithms and their design to develop most of the software systems. Earlier algorithms have been developed for music retrieval [8], [9] and composition [10], [11] but researchers are still figuring out ways to find better synergies between the two domains. In this paper, a correlation has been attempted. Based on one simple strategy that musicians use, we show how algorithmic thinkers and data structure designers apply it. We cannot say if they had this idea in their minds but this analogy shows how two disciplines can be convergent. It is not restricted to what we discuss in the paper but many such analogies can be drawn.

There is also a possibility of improving the retrograde strategy by applying the knowledge of computer science. This makes the interaction between the two domains mutually beneficial. We discuss this as we understand how this is applied in context of the *stack* data structure. The difficulties that we might face in developing the analogy, can be turned into opportunities to advance the technique.

We do not claim that learning music is easier than learning algorithms or vice-versa but here we present a way to simplify understanding at the conceptual level. Both the domains are equally involved but a correlation always stimulates an interest towards the other domain. Thus, this learning technique aims at interest generation and conceptual enhancement in the presented data structures and music. The approach is simple and easy to understand and teach in classroom time. Both the strategy and its understanding can be delivered to a student having no prerequisite knowledge of music. We describe the Retrograde Strategy in Section 2 followed by some similarities and differences in Section 3. The evaluation and results are mentioned in Section 4 with conclusions and future directions in Section 5.

II. The Retrograde Strategy

In this section, we first discuss the retrograde strategy in detail and establish synergy between the two domains. Retrograde strategy has been widely used in medical domain [12]–[14]. However, we limit ourselves to its use in music composition. Retrograde when defined in terms of music means to play the notes in reverse order. When we have a basic idea of the melody at hand, we derive a new tune by mirroring the notes. To understand it better, let us say you have one followed by two in a particular melody, we append two followed by one to get a new tune. To get the musical feel, you can map these two numbers with intensity of the same note. We get into the music composition domain to explain it

2161-377X/18/$31.00 ©2018 IEEE
DOI 10.1109/ICALT.2018.00066

further. We take a work from Wikipedia that shows retrograde inversion to illustrate what it is. The image displayed in Figure 1 is an example of retrograde transformation. The image might not give an insight at the first look but if we observe the first(A) and fourth(A'), second(B) and fifth(B') and also third(C) and sixth(C') bars carefully, what we see is the mirror image of the notes. The immediate observation that we make at this point is that the last note to be played, is the first one to be played again if we use retrograde strategy. Looking at this a programmer can immediately think of a data structure in which the value which enters at the last is the first to be retrieved. One more idea is of the recursive calls which keep running until the end is reached and are followed by backtracking. We explain these concepts in the next subsections so that the analogy is clearly brought out.

A. Recursion and Algorithm Design through Retrograde Strategy

Recursion is a common technique used in divide and conquer algorithms and is a core component of STEM education today [15]. It was studied and published by Freud et al. in 1976 [16]. Recursion is also at the heart of many of the commonly used algorithms such as *quicksort* and *depth first search*. In recursion what we observe is that as a function calls itself, the value of the parameter(if any) changes so that finally a base condition is reached. Upon reaching this condition, we then begin to backtrack and finally the last value of the function is returned. Researchers have tried to explain recursion in different simple ways [17]–[19]. We present one more simple way in this section to understand its basics. Suppose that every function call is a note that is played. As long as you reach the base condition, a note is being played. After that when the functions start returning and we backtrack, we play the same note again. This forms a retrograde of the musical notes that we played before reaching the base condition. In this way we apply the retrograde strategy in the context of music and recursion. To illustrate the same with an example we look at the first few programs that we learn in recursion: finding the factorial of a number and computing the fibonacci series.

[1]Machaut, the opening and closing bars of 'ma fin est mon commencement' (My end is my beginning) by Guillaume de Machaut

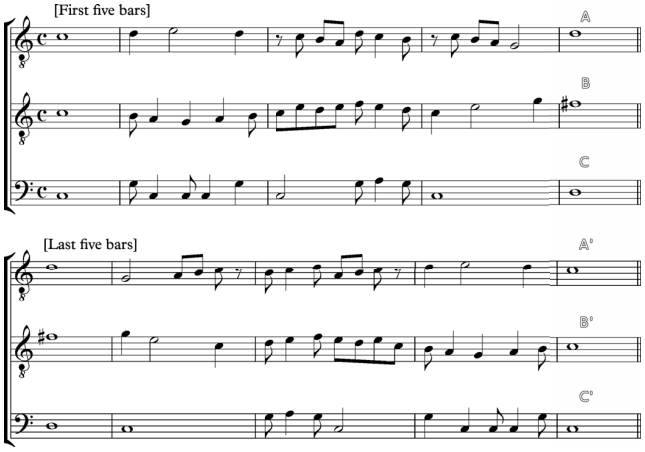

Fig. 1. Retrograde strategy to compose music[1]

1) **Factorial:** In order to find the factorial of a number, we can either compose a program using loops or using recursion. When we use recursion to find the factorial of a number, we define a function fact(n) which we mention below:

```
int fact(int n) {
        if (n == 1)
                return 1;
        else
                return (n * fact(n-1));
        }
```

The code itself is not that intuitive but when we look at the control flow in Figure 2, we can see how it can be co-related with retrograde strategy. If we play this as a music notes, a series of mapping from fact(5) to fact(1) with different notes, we get the retrograde of a pentatonic scale. This example was straight forward so now we look at a better example that is the Fibonacci series in the next part. In the same case, if we observe for the factorial of a large number, the recursion tree grows and so does the complexity of the algorithm. However, we can map it by repeating the notes in pairs. The mapping that we will obtain in this case will also be following the retrograde pattern. To verify this, we can understand it by considering the following two cases:

   a) The number of recursive calls is odd: In this case, as per our suggested technique, the last note will have two same notes for instance DO, DO. Therefore, when we retrograde or retrace the path, the DO note will be mirrored with the other and remaining all other notes follow the same pattern.
   b) The number of recursive calls is even: This will be an obvious case as the last call will map on to itself.

2) **Fibonacci Series:** To compute an element of this series, we can also use a loop based approach but we focus on the recursive method. An explanation of this can be found in the work by Roberts et al. [20] and also in works by Ivan et a. [21]. Recursive functions are difficult for a novice programmer to understand. We define the function to compute the n^{th} number of the Fibonacci series as fib(n). We start from 0 so if we say fib(3) it means that we are talking of the 4^{th} number in the series.

```
int fib(int n) {
        if (n == 0)
                return 0;
        if (n == 1)
                return 1;
        else
                return
```

```
                  fact(5)            enter into fact(5)
                     |
                     V
                5 * fact(4)          enter into fact(4)
                     |
                     V
                4 * fact(3)          enter into fact(3)
                     |
                     V
                3 * fact(2)          enter into fact(2)
                     |
                     V
                2 * fact(1)          enter into fact(1)
---------------------------------------------------------
return from fact(1)  1
                     |
                     V
return from fact(2)  2 * (1)
                     |
                     V
return from fact(3)  3 * (2 * 1)
                     |
                     V
return from fact(4)  4 * (3 * 2 * 1)
                     |
                     V
return from fact(5)  5 * (4 * 3 * 2 * 1)
```

Fig. 2. Control flow in the defined function fact(n)

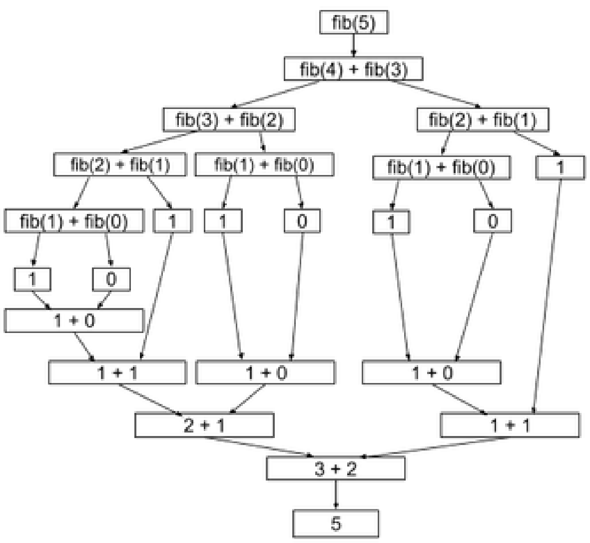

Fig. 3. Control flow in the defined function fib(n)

```
                (fib(n-1) + fib(n-2));
}
```

The control flow is illustrated in Figure 3. It might be difficult to comprehend this at first as there are multiple places where the retrograde strategy needs to be applied to understand better. To simplify this task, arrows are used. If we look at any one of these arrows and try to trace back, what we get is a retrograde of a particular set of notes. Say we follow the longest path that is:

$$fib(5) \rightarrow fib(4) \rightarrow fib(3) \rightarrow fib(2) \rightarrow fib(1)$$

Then the retrograde of this will be:

$$1 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 5$$

When we say fib(4) and the number 3, they are taken to be the same note as fib(4) = 3. In this way for each of these values of fib(n), for different n values if we define different notes and play them again while backtracking, we get the retrograde of the notes. Similarly, this can be done by tracing all the arrows. A horizontal line in between can be used as mirror to traceback the values. This is an example where multiple retrogrades are taking place simultaneously, like different musical instruments playing the set of notes with a small difference in time interval. This shows how musical a piece of code could be. If we want to reduce the number of instruments and switch to just one, like we did in the previous case, we must use a lookup table and store the values once computed. When we do this, we just get the retrograde inversion of the longest series. All the functions are called only once for a particular value of n.
Here, the strategy might seem to have broken down as we just look at one path and try to retrace it though we can do the same for all the paths. In this case, we can think of enhancement of the retrograde strategy by allowing multiple nodes to flow through simultaneously, each returning back after a set number of time frames. We can model the entire call to a list with a few empty entries for short recursive paths to ensure that all are of same length.

B. Comparable Data Structure [2]

The property which makes stack different from other similar data structures is LIFO (Last In First Out). This means that the value that is entered at the end is the first one to be retrieved. We have a pointer called top which stays at the last entered value and is NULL if the stack is empty. We also define two basic operations on a stack:

1) **Push(k):** This means to increment top and then add an element k at the top of the stack.
2) **Pop():** This means to remove the element pointed by the top of the stack.

Now suppose we have a set of numbers to be pushed on top of the stack say 1, 2, 3 and 5. As a new element comes, the top is incremented and the element is placed where the top points. After inserting all the four elements, if we execute a pop function, the value that is removed is the last one that was entered which is 5 in this case. This scenario is depicted in **Figure 4**. Similarly if we perform more pop operations on the stack, we get values 3, 2 and 1 in order. Now this can be compared to the retrograde of the notes in **Figure 5**. This example clearly demonstrates the synergy.

However, this might not seem quite obvious when the push and pop operations occur together. In the case mentioned above, the pop operation is carried out until the stack is empty. Suppose not, we will have several small notes where retrograde is being carried out. As we converge them and filter them out, even the last elements left form an image on each other to give another retrograde. This is the point where the data structure is a means to extend this idea. Suppose we push n elements and pop k elements where $k < n$, we get a retrograde of k notes. Similarly, we get a retrograde of p notes if we pop p elements from $n - k$ elements where $p < n - k$. This is an example of a complete song which is composed using retrograde strategy at several instances and also on the overall song. Suppose at the end, the final operations are done on some m elements, then those m elements also form a retrograde. In this way, we show that the idea is complete and an analogy can be drawn between the retrograde strategy used in music and the similar uses in data structures and algorithms.

III. Other Similarities and Differences

In this section, we give points of intersection and the variations between the two domains. We define two labels for each point as **S** and **D** where S will represent that this is a point of similarity and D shows that it is a point of difference.

S: Both follow logical rules

Composing music requires Trivium just like programming does. Trivium is defined by ancient Greeks and needs:

1) Grammar: This can be musical notations or syntax of programming languages.
2) Logic : To get code or music to work together logic is needed.
3) Rhetoric: The music and code must be appealing and understood by its audience and users respectively.

[2]Generally, recursion is implemented using a stack data structure.

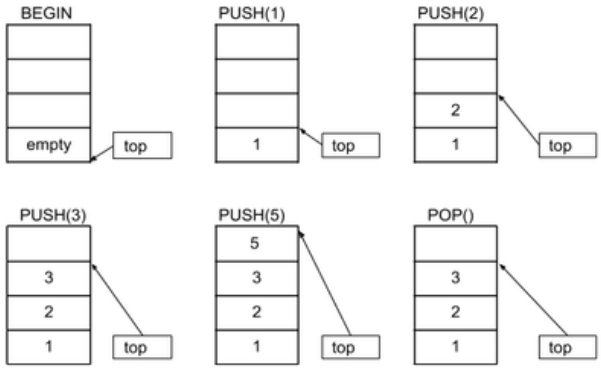

Fig. 4. Four push and one pop operation on a stack

S: Both can be collaborative or individualistic Writing a code into small pieces and together combining them is common practice for developing a software. Similarly a music can be composed by a logical fusion of individual pieces by different artists. However, both music and code composition express ideas of a person and no one stops a single programmer from writing the entire code all by himself or a musician from composing a melody alone.

D: Programming is for solving a problem The main purpose behind writing a program is to address a particular problem. On the contrary, music is purely based on expressing the emotions of individuals. It might aid the composer or listener but there is no definition of a particular problem it might address. This draws a clear line between the domains.

IV. Evaluation

In order to evaluate, we used retrograde strategy to teach recursion in *Computational Engineering* course offered in the first semester at an *Institute of Technology*. First the students were taught recursion and we demonstrated the normal control flow and stack. An observation made was that students often got confused while they dry run the code in retracing the path of recursion. Later, the control flow was demonstrated using strategy with the use of an example: the name of a musical note was assigned to each of the function calls. This later helped in retracing the path by singing the notes backwards. We recorded their views in a feedback form. Though the sample space was limited to thirty students which is the total strength of Computer Science first year students at the institute, the results obtained were promising. About 80.6% of the class found the analogy helpful and 81.2% said that they preferred this over the usual way. In the following section, we discuss the similarities and differences between the two domains. Each strategy is first explained and later on the application in computer science is demonstrated. A game was developed along the same lines just to create some interest named Retrograde Simon: a version of the traditional Simon game which asks you to remember a series and then repeat the

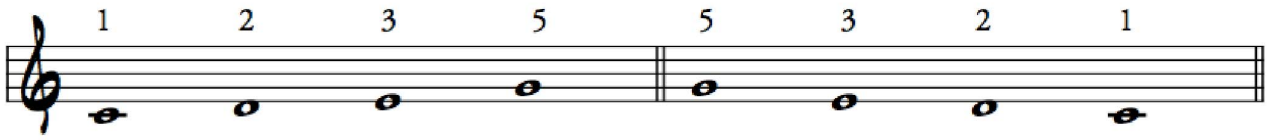

Fig. 5. Four notes and the retrograde

same from your memory. *The Retrograde Simon* does the same for musical notes with a difference that you need to trace back rather than memorizing the same sequence. It was developed using Phaser [22] provided by MIT.

V. Conclusion and Future Directions

No matter how many similarities we are able to draw, there will be some obvious differences between the two domains. The idea behind this paper is not to claim that the two domains are same but to bring out the areas in which each can benefit from their synergy. There are points where the two domains intersect and it is at that point where we get to improve upon the existing by studying the analogy. Even if improvement is not possible, it is always possible to explain concepts in one domain with the help of concepts in some other domain. This is what we established in this paper. We just touched upon one strategy used in music, similarly there can be many such strategies which can be analogous to one in a different domain. Studying more such correlations makes it possible to teach one domain using another. Some concrete future directions are:

- *Composition of music and Composition of Code* - Changing the instruments and varying the notes creates a different version of the music. Can we learn from this to compose code from open source components?
- *Classification of Music and Code* - A closely related area is the classification of music based on *Ragas*, *Genres*, *instruments*, *mood* and so on. Can we classify code based on metrics and multiple dimensions based on the techniques in music?
- *Music Clones and Code Clones* - Music is a creative field yet reuses existing compositions for generating new ones. Can we learn from music clones and improve our code clone techniques?
- *Tools and Techniques* - Music industry uses a set of standardized extensive tools that can anlayze the music from several perspectives. How do we develop a sthem for computer science?
- *Gamification, Music and Code* - Can we integrate gamification, music and code composition techniques?
- *Parallel Playing* - There are different control flows merging into one if there are multiple recursive calls from same function. Can different instruments signifying different flows be used to make a melody?

References

[1] D. R. Hofstadter, *Gödel, escher, bach*. Vintage Books New York, 1980.

[2] M. Papastergiou, "Digital game-based learning in high school computer science education: Impact on educational effectiveness and student motivation," *Computers & Education*, vol. 52, no. 1, pp. 1–12, 2009.

[3] M. Overmars, "Teaching computer science through game design," *Computer*, vol. 37, no. 4, pp. 81–83, 2004.

[4] L. Cao, V. Gorodetsky, and P. A. Mitkas, "Agent mining: The synergy of agents and data mining," *IEEE Intelligent Systems*, vol. 24, no. 3, 2009.

[5] S. Chandrasekaran, G. Silver, J. A. Miller, J. Cardoso, and A. P. Sheth, "Web service technologies and their synergy with simulation," in *Simulation Conference, 2002. Proceedings of the Winter*, vol. 1. IEEE, 2002, pp. 606–615.

[6] F. Martin-Sanchez, I. Iakovidis, S. Nørager, V. Maojo, P. de Groen, J. Van der Lei, T. Jones, K. Abraham-Fuchs, R. Apweiler, A. Babic *et al.*, "Synergy between medical informatics and bioinformatics: facilitating genomic medicine for future health care," *Journal of biomedical informatics*, vol. 37, no. 1, pp. 30–42, 2004.

[7] Z. W. Geem, *Music-inspired harmony search algorithm: theory and applications*. Springer, 2009, vol. 191.

[8] N. Hu, R. B. Dannenberg, and G. Tzanetakis, "Polyphonic audio matching and alignment for music retrieval," in *Applications of Signal Processing to Audio and Acoustics, 2003 IEEE Workshop on.* IEEE, 2003, pp. 185–188.

[9] A. Guo and H. T. Siegelmann, "Time-warped longest common subsequence algorithm for music retrieval." in *ISMIR*, 2004.

[10] R. De Prisco and R. Zaccagnino, "An evolutionary music composer algorithm for bass harmonization," *Applications of evolutionary computing*, pp. 567–572, 2009.

[11] F. Taga, "Smart music algorithm for doa estimation," *Electronics Letters*, vol. 33, no. 3, pp. 190–191, 1997.

[12] S. Saito, "Different strategies of retrograde approach in coronary angioplasty for chronic total occlusion," *Catheterization and Cardiovascular Interventions*, vol. 71, no. 1, pp. 8–19, 2008.

[13] B. K. Kaspar, D. Erickson, D. Schaffer, L. Hinh, F. H. Gage, and D. A. Peterson, "Targeted retrograde gene delivery for neuronal protection," *Molecular Therapy*, vol. 5, no. 1, pp. 50–56, 2002.

[14] T. Powley, E. Fox, and H. Berthoud, "Retrograde tracer technique for assessment of selective and total subdiaphragmatic vagotomies," *American Journal of Physiology-Regulatory, Integrative and Comparative Physiology*, vol. 253, no. 2, pp. R361–R370, 1987.

[15] B. R. Belland, A. E. Walker, N. J. Kim, and M. Lefler, "Synthesizing results from empirical research on computer-based scaffolding in stem education: A meta-analysis," *Review of Educational Research*, vol. 87, no. 2, pp. 309–344, 2017.

[16] G. Freud, "On the coefficients in the recursion formulae of orthogonal polynomials," in *Proceedings of the Royal Irish Academy. Section A: Mathematical and Physical Sciences*. JSTOR, 1976, pp. 1–6.

[17] W. Dann, S. Cooper, and R. Pausch, "Using visualization to teach novices recursion," *ACM SIGCSE Bulletin*, vol. 33, no. 3, pp. 109–112, 2001.

[18] D. Wilcocks and I. Sanders, "Animating recursion as an aid to instruction," *Computers & Education*, vol. 23, no. 3, pp. 221–226, 1994.

[19] R. Sooriamurthi, "Problems in comprehending recursion and suggested solutions," in *ACM SIGCSE Bulletin*, vol. 33, no. 3. ACM, 2001, pp. 25–28.

[20] E. Roberts and E. Roberts, *Thinking recursively*. J. Wiley, 1986.

[21] I. Stojmenovic, "Recursive algorithms in computer science courses: Fibonacci numbers and binomial coefficients," *IEEE Transactions on Education*, vol. 43, no. 3, pp. 273–276, 2000.

[22] T. Faas, *An Introduction to HTML5 Game Development with Phaser. js*. CRC Press, 2017.