Dynamic Chord isdentification using Finite State Machines

Sripriya Konjarla
Department of Computer Science and Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4cse22121@bl.amrita.edu

Likhitha Sri
Department of Computer Science and Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4cse22125@bl.amrita.edu

Ajay Ratnam
Department of Computer Science and Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4cse22146@bl.amrita.edu

Srujan Reddy
Department of Computer Science and Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4cse22151@blr.amrita.edu

***Abstract*—The Music Analyzer project is a musician-friendly tool that improves chord analysis and identification. It uses finite automata and Python to process MIDI files created from BandLab recordings in an efficient manner. By utilizing Finite State Machines, it quickly recognizes chords on input, and enables musicians to easily learn and play along by breaking down key signatures and creating major and minor scales. The system's user-friendly interface promotes the mastery of complicated compositions by musicians by offering clear outputs. This software makes music analysis easier, making it suitable for both practice and performance, enhancing the musical experience.**

***Index Terms*—FSM, Music analysis, MIDI, Chords, Scales, Queue, Keys, Octave, Notes, Transition, Threshold, States, Sort, Press, Detection, Parse, Validate**

I. Introduction

We present a novel approach to music analysis through a user-friendly interface. The Music Analyzer employs a simple yet powerful system where users input musical intervals, such as "C, E, G," and the program interprets and plays the corresponding chord type. This innovative tool accommodates a wide range of chord qualities including major, minor, augmented, diminished, dominant, and more.

The program utilizes finite state machines to swiftly identify the input intervals, analyze their harmonic structure, and generate the appropriate chordal output in real-time. Through its intuitive design, the Music Analyzer offers musicians, composers, and enthusiasts an efficient means to explore, understand, and experiment with various chord qualities, fostering a deeper engagement with musical composition and theory.

The Music Analyzer is designed to be versatile, allowing users to input any combination of notes and receive instant feedback on the resulting chord type. This capability not only aids in quick chord identification but also serves as an educational tool for beginners learning music theory and composition.

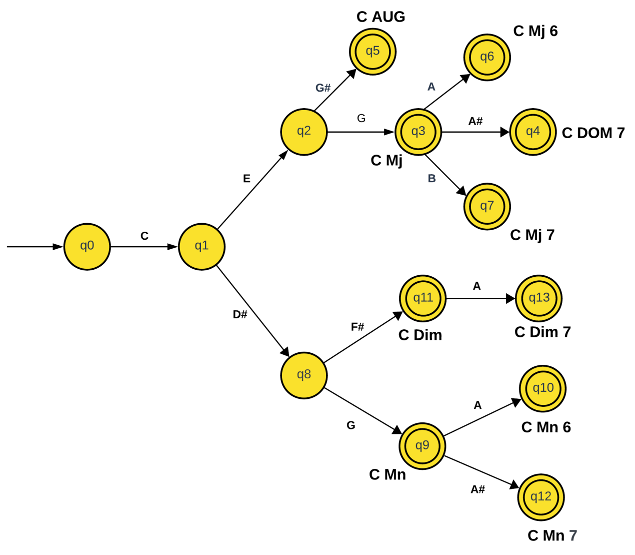


Fig. 1. NFA for C Chord

II. Literature Survey

Music analysis is a multifaceted field encompassing various techniques for dissecting and understanding musical structures. Existing approaches leverage diverse methodologies, including deep learning architectures [1], statistical analysis [2], and symbolic music representation [3].

Among these, symbolic music representation offers a compelling approach for music analysis tasks owing to its ability to capture the inherent structure and relationships within musical pieces [4], [5]. This method employs a symbolic representation of music, such as MIDI files, which encode musical elements like pitch, duration, and tempo. By leveraging symbolic representations, music analysis systems can effectively analyze and interpret musical content.

The study presented in [6] proposes a novel approach for generating alapana, an improvisational form of Carnatic music, using Finite State Machines (FSMs) and Generative Adversarial Networks (GANs). This demonstrates the potential of integrating FSMs with other advanced techniques for music generation. Additionally, the design of the Super Mario game using FSMs showcases their application in game design and behavior modeling [7].

Research in [8], [9] explores the implementation of FSMs in digital systems, highlighting their efficiency in handling sequential logic, which is relevant to processing musical sequences. Speech recognition systems using Weighted Finite-State Transducers further illustrate the versatility of FSMs in handling complex sequential data [10].

Furthermore, the work in [11] uses the Holt-Winter Model to predict note events in music, showcasing the application of statistical models for understanding musical patterns.

Other approaches, such as the one described in [12], analyze musical structure using information rate, while [13] incorporates beat and key detection for automatic chord detection. These studies demonstrate the diversity of techniques used in music analysis.

Comparative evaluations of decomposition methods based on pitch estimation of piano notes highlight the effectiveness of different signal processing tools in music analysis. For example, variational mode decomposition has been found to perform better across various octaves compared to wavelet and dynamic mode decomposition methods [14].

Our proposed Music Analyzer builds upon the foundation of symbolic music representation and FSMs to provide a real-time and user-friendly tool for music chord identification and analysis [15]. By combining these techniques, we aim to offer a versatile and efficient system that caters to the needs of musicians and music enthusiasts.

III. Methodology

A. FSM Strategy

The core aim of the system is to efficiently identify the keys which are being pressed together (which can be in any order real-time), so that we can identify the chord being played, at the same time, we should be able to ignore the keys which are not part of the chord. We used finite state automata to identify chords, each note causes transition in the state, and the final state shows the specific chord which is detected. The figure 1 shows the NFA for the different chords of C scale.

B. Proposed Algorithm

- FSM: We first designed the DFA used to build the FSM, which has all the transitions to identify the chords.
- Multiple Octaves: Sometimes the note in the chord overlaps into another octave, so we have multiple FSM's parallely processing the key presses from the two octaves.
- Parallel Key Press: Usually while playing, musicians won't press the keys of the chord at the "exact" same time, so we recorded the key presses during a specified THRESHOLD, and then sort them to process them.
- Invalid Key's: If a pressed key is not part of the identifying chord, then we reset and transition it back to the start state.

C. Finite State Machine (FSM)

The FSM is the core component responsible for chord detection. It maintains states, and transitions, and handles chord detection logic. The FSM consists of the following key elements:

- States: Defined with transition rules and whether they are final states.
- Start State: The initial state of the FSM.
- End States: States indicating the end of a chord sequence.
- Detected Queue: Stores detected chords for a certain duration.
- Lock: Ensures thread safety when accessing the detected queue.

D. Recording

- Direct: Plug and play, the user can directly plug their musical instrument which supports MIDI, and our program will detect the played chords.
- Recorded: The user can also record it on applications like Bandlab Studio which is more sophisticated and the recording can be exported to a MIDI file, which can be read by our program. Figure 3 and 5 show an example of a recording we made, which is a screenshot of the MIDI mapping for the song.

E. Used Technology

We used a custom-made FSM, to handle all these key presses, and identify the chord, based on our needs. As a result, we have come up with a DFA comprising of 157 states and 1877 transitions. Figure 2 shows the control flow of the program.

F. Program Overview

The program consists of several Python functions and classes to process musical notes and detect chords. It also utilizes threads for concurrent executions.

G. Key Components

- Sorter: This function is used for sorting musical notes based on their pitch.
- Get Note: Extracts the musical note and its octave from the input.
- Press: Simulates pressing a musical key and triggers the Finite State Machines (FSMs) accordingly.
- State: Represents a state in the FSM, with transition rules defined.
- FSM: Implements the Finite State Machine for chord detection. It maintains states, transitions, and a queue for detected chords.

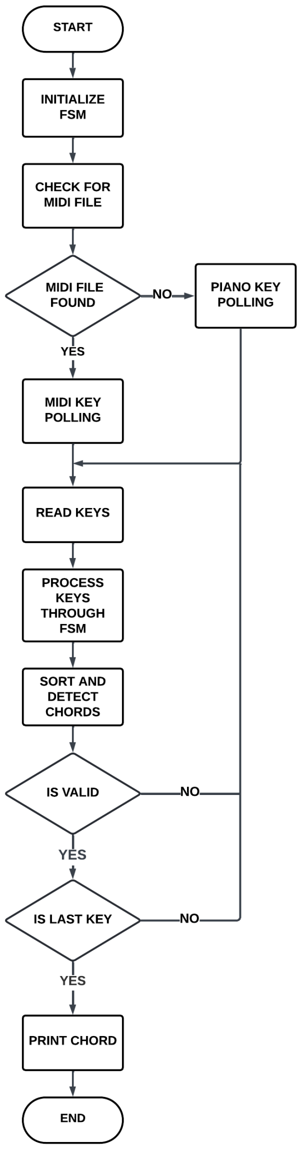


Fig. 2. Flowchart of Program

H. Execution Flow

The program begins by reading transition rules and state definitions from an external file, which are used to initialize the FSM states. Multiple FSM instances are then created to process different octaves of musical notes concurrently. When a musical note input is received, the program simulates a key press and triggers the FSM processing. Each FSM processes the input, transitions between states, and detects chords accordingly.

IV. Result

In our tests, the Music Analyzer did really well at figuring out chords from MIDI input. We tried playing many different chords and ways, and it always gave quick and accurate results. Figure 4 and 6 show the results of the recording we used to verify the program. The system uses Finite State Machines, which helps it work in real-time and gives fast feedback. Overall, our tests show that the Music Analyzer is great for helping people get better at playing music and understand how it all works. It's a handy tool for musicians and anyone interested in music!

The code for our Music Analyzer is available on GitHub: https://github.com/ajratnam/chord-analyzer-fsm.

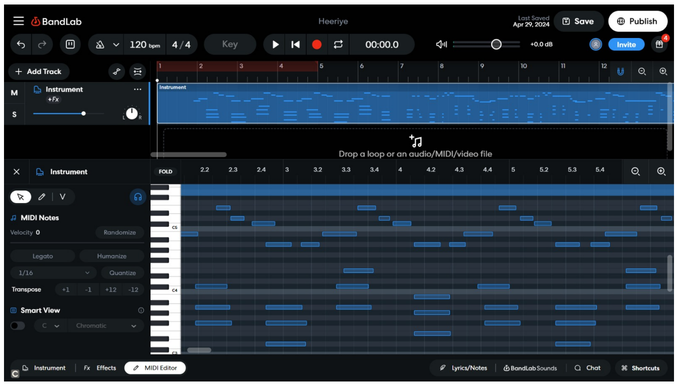


Fig. 3. Recording on Bandlab Studio

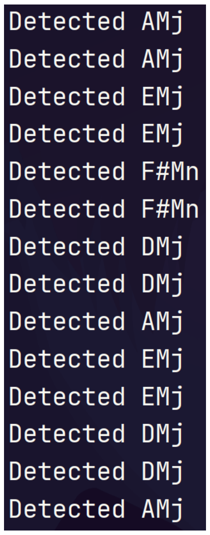

```
Detected AMj
Detected AMj
Detected EMj
Detected EMj
Detected F#Mn
Detected F#Mn
Detected DMj
Detected DMj
Detected AMj
Detected EMj
Detected EMj
Detected DMj
Detected DMj
Detected AMj
```

Fig. 4. Detected chords from our Song

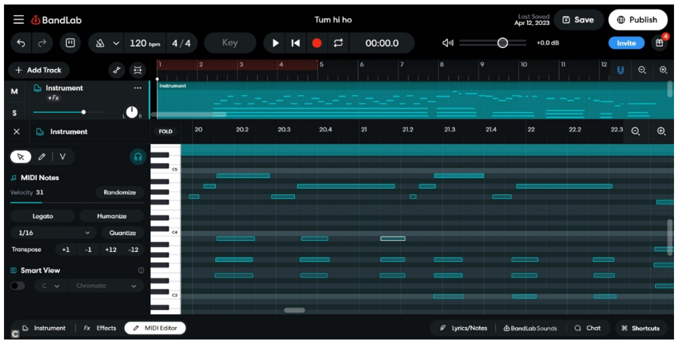
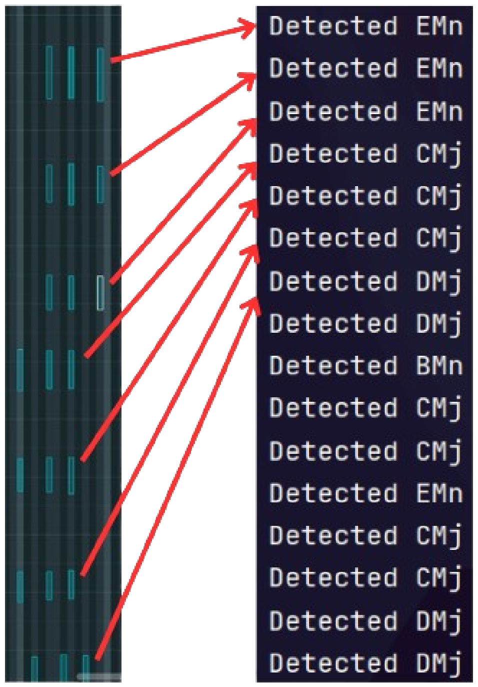


Fig. 5. Another Song on Bandlab Studio



```
Detected EMn
Detected EMn
Detected EMn
Detected CMj
Detected CMj
Detected CMj
Detected DMj
Detected DMj
Detected BMn
Detected CMj
Detected CMj
Detected EMn
Detected CMj
Detected CMj
Detected DMj
Detected DMj
```

Fig. 6. Verifying output with notes played

V. Conclusion

Utilizing Finite State Machines (FSMs) innovatively, the Music Analyzer presents a robust solution for the identification and analysis of chords. Capable of handling multiple simultaneous key presses, this tool empowers users to enhance their piano skills with ease and precision. By leveraging FSMs, we have created a versatile and efficient system that promotes deeper engagement with music theory and composition. With its user-friendly interface and real-time feedback, the Music Analyzer offers a valuable resource for musicians, composers, and enthusiasts alike, enriching the musical experience and facilitating continuous growth and exploration in the realm of music.

References

[1] D. Das and M. Choudhury, “Finite state models for generation of hindustani classical music.”

[2] J. Park, K. Choi, S. Jeon, D. Kim, and J. Park, “A bi-directional transformer for musical chord recognition,” 2019.

[3] J. Pinnamaneni, “Theory of computation music automata,” 04 2019.

[4] S. Adhikary, M. S. M, S. S. K, S. Bhat, and K. P. L, “Automatic music generation of indian classical music based on raga,” in *2023 IEEE 8th International Conference for Convergence in Technology (I2CT)*, 2023, pp. 1–7.

[5] J. P. Forsyth, “Automatic musical accompaniment using finite state machines.”

[6] V. Jayatharan and D. Alwis, “Alapana generation using finite state machines and generative adversarial networks,” in *2023 International Research Conference on Smart Computing and Systems Engineering (SCSE)*, vol. 6, June 2023, pp. 1–6.

[7] A. S. Nambiar, K. Likhita, K. V. S. Sri Pujya, and M. Supriya, “Design of super mario game using finite state machines,” in *Computer Networks and Inventive Communication Technologies*, S. Smys, P. Lafata, R. Palanisamy, and K. A. Kamel, Eds. Singapore: Springer Nature Singapore, 2023, pp. 739–752.

[8] M. Kubica and D. Kania, “Technology mapping of fsm oriented to lut-based fpga,” vol. 10, no. 11, 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/11/3926

[9] J. van Gurp and J. Bosch, “On the implementation of finite state machines,” 01 1999.

[10] D. Biswas, S. Nadipalli, B. Sneha, and M. Supriya, “Speech recognition using weighted finite-state transducers,” in *2022 IEEE 7th International conference for Convergence in Technology (I2CT)*, 2022, pp. 1–5.

[11] E. Feldman, “Midi music model,” in *SoutheastCon 2015*, April 2015, pp. 1–4.

[12] S. Dubnov, “Analysis of musical structure in audio and midi signals using information rate,” 01 2006.

[13] V. Zenz and A. Rauber, “Automatic chord detection incorporating beat and key detection,” in *2007 IEEE International Conference on Signal Processing and Communications*, Nov 2007, pp. 1175–1178.

[14] U. Vamsi Krishna, R. Priyamvada, G. Jyothish Lal, V. Sowmya, and K. P. Soman, “A comparative evaluation of decomposition methods based on pitch estimation of piano notes,” in *Smart Computing Techniques and Applications*, S. C. Satapathy, V. Bhateja, M. N. Favorskaya, and T. Adilakshmi, Eds. Singapore: Springer Singapore, 2021, pp. 833–843.

[15] C. F. Laurent Oudre, Yves Grenier, “Template-based chord recognition : Influence of the chord types.”

[16] A. Nambiar, K. Likhita, P. Kothapalli, and N. Panda, “Exploring the power of deep learning for seamless background audio generation in videos,” 07 2023, pp. 1–7.

[17] N. Panda and M. Supriya, “Blackhole attack impact analysis on low power lossy networks,” in *2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT)*, Oct 2022, pp. 1–5.