EECS 510: Theory of Computation

The 12 Bar Blues and Nondeterministic Finite Automaton

Carlos Mbendera

May 8, 2025

1. Introduction

This past semester, I have had the pleasure of taking EECS 510 under the tutelage of Dr. Jennifer Lohoefener and for my final project, I have opted to represent the 12 Bar Blues chord progression as a Nondeterministic Finite Automaton (NFA).

First, allow me to provide context on what a 12 Bar Blues is and how our language represents it. Alongside that, I shall also highlight a prior but failed attempt to represent the same language with a Puash Down Automaton (PDA).

In music theory, we group the beats of a song into bars. A beat is essentially a unit of measurement for music and rhythm, whenever you are listening to a song and tapping your foot along to the drummer or chords. A single tap of your foot is a single beat. Most popular and standard songs use a 4/4 system, this means 4 beats make one bar. More complex forms of music such as Jazz can, and often, use strange magical timings that are not 4/4, but for this project we are not concerned with other factors that can affect our bars such as BPM, musical instrument notation, timings and more fun things.

In the American genre known as Blues, there exists a popular song structure known as a "12 Bar Blues." You can think of it as a standard and predefined framework for how songs can be written and performed once a Musical Key has been chosen. Hence, it allows musicians and listeners to quickly grasp what to expect if the topic comes up. For example, if a jam group of amateur musicians convenes for the first time and they want to perform a song together with little to no preparation, the leader may say, "let's do a 12 bar blues in they key of C Major." This works because the musicians would know that a 12 Bar Blues follows the following Chord Progression:

- I chord for 4 bars
- IV chord for 2 bars
- I chord for 2 bars
- V chord for 1 bar
- IV chord for 1 bar
- I chord for 1 bar
- V chord for 1 bar

In music theory, I represents the first Major Chord in a scale, IV the fourth major chord and V is the fifth Major Chord.

If we were to write this as a string, then the 12 Bar Blues is I-I-I-I-IV-IV-I-I-V–IV-I-V, with the dashes representing λ transitions or an optional melody transition done in between chords, which I shall label as N from now on.

Moreover, for simplicity sake and readability, I shall write I as 1, IV as 4 and V as 5. So, our 12 Bar Blues can be written as 1N1N1N1N4N4N1N1N5N4N1N5 or 111144115415.

This is the most basic structure of a string in our language, a single 12 Bar Blues with no improvisations during transitions and is not repeated, as in we don't perform 24, 36 or 48 bars.

In music theory, we represent the musical notes that build up a melody with letters ranging from A through G. In my language, N is valid as long as it's a member of the set {a,A,b,c,C,d,D,e,f,F,g,G}*.

The set contains both capital and small letters to handle the Sharps and Flats of musical notes. So C is c Sharp, however, it is important to note that we don't have an explicit Flat in our notation because A Sharp is the same as b flat, so to avoid repetition we only show the sharps and not the flats. Additionally, we don't have E and B because music theory states that e and b don't have sharps.

Lastly, despite being named a 12 Bars, we're not actually capped at performing 12 Bars, we can do multiple iterations of these 12 bars. Hence, the language accepts any valid series of 12 Bar Blues, so a 120 Bar string is accepted as long as it follows the other rules. We don't accept the empty string as the 12 Bars Blues requires a minimum of 12 bars.

2. Grammar

Based on these rules and understanding of how a 12 Bar Blues string should be generated, we can come up with the following rules.

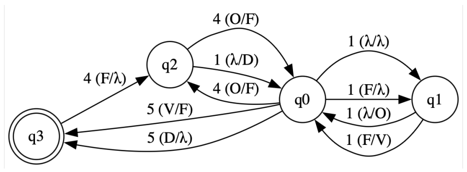
$$
\begin{aligned}
S &\to 1N1N1N1N4N4N1N1N5N4N1N5E \\
N &\to \text{base}\,N \mid \text{sharp}\,N \mid \lambda \\
\text{base} &\to a \mid b \mid c \mid d \mid e \mid f \mid g \\
\text{sharp} &\to A \mid C \mid D \mid F \mid G \\
E &\to S \mid \lambda
\end{aligned}
$$

3. Automaton

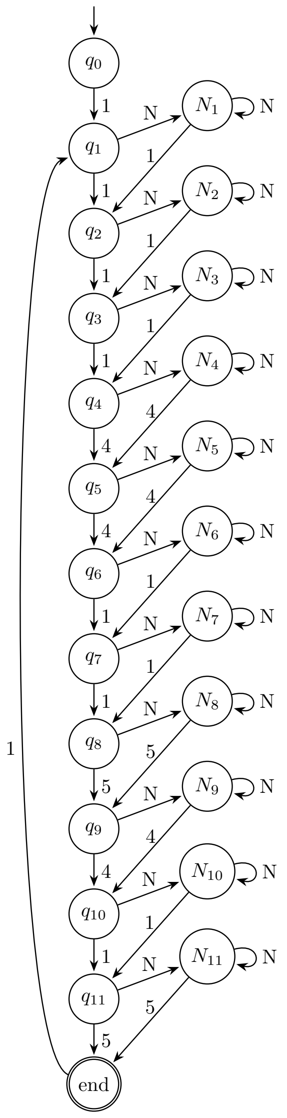
Initially, I created a PDA for this language as we have a significant amount of repetition as the 12 Bars only has 3 Chords: 1,4 and 5.

Moreover, we can abstract our notes as N for readability. Hence, I was certain we could create a visually and aesthetically pleasing PDA to minimize repetition. Unfortunately, the one I made did not pass the basic test case of no reparations and no melody transitions, thus causing me to switch to an NFA as it was easier to follow and comprehend, albeit a little basic. Below you can see my failed automaton.

The idea was that reading 1 pushed O, 4 pushed F and 5 pushed V. We would maintain the progression order by popping the expected previous chord.



Below is the final NFA I settled on, as it is easier to follow along. Note N symbol is interchangeable with the actual notes. I use it to save space with certain transitions. So N is the same as ($a \mid b \mid c \mid d \mid e \mid f \mid g \mid A \mid C \mid D \mid F \mid G \mid \lambda$).



4. Data Structure

This is the data structure of our automaton.

Please note that I have used N instead of it's many symbols ($a \mid b \mid c \mid d \mid e \mid f \mid g \mid A \mid C \mid D \mid F \mid G \mid \lambda$) for convenience's sake but please treat them the same.

Recall from the introduction that a 12 Bar Blues follows this path (chord progression) and may or may not have melody transitions (N) in between chords.

- 1/I chord for 4 bars
- 4/IV chord for 2 bars
- 1/I chord for 2 bars
- 5/V chord for 1 bar
- 4/IV chord for 1 bar
- 1/I chord for 1 bar
- 5/V chord for 1 bar

The semantics for our Data Structure are therefore:

- Each of the 12 Chords are represented by states q1 through end. So q1→ q4 are the 1 chords, q5 and q6 are the 4 Chords and so on.
- We start at q0 and once we see our first chord represented by 1, we are in the state of chord 1 (q1).
- At this point we can either have a melody transition, by reading N and entering N1 or go to the next chord.
- If we read N and go to N1, then we self loop in N1 as long as we have a valid N character. Once we read the next chord, we enter the next and only valid next chord state.
- In this example, it would be q1→ q2 via a 1. So if I went through N it would be q1→ N1→q2 by reading 1 at the N1→q2 transition.
- Once we reach the end State, if we've processed the entire string, we stop. Otherwise, if we have a 1 then that means the 12 bar blues is starting again and we should transition to q1 and go through all 12 chords again.
- This is my biased music taste showing but I do not allow the language to have melody transitions when we're restarting the 12 bar blues, hence the lack of a suffix N for the last 5 Note and no Nend state.

On the next page, you can observe a data structure of our automata

```
q0 q1 q2 q3 q4 q5 q6 q7 q8 q9 q10 q11 N1 N2 N3 N4 N5 N6 N7 N8 N9 N10 N11 end
1 4 5 N
q0
end
q0 1 q1
q1 1 q2
q2 1 q3
q3 1 q4
q4 4 q5
q5 4 q6
q6 1 q7
q7 1 q8
q8 5 q9
q9 4 q10
q10 1 q11
q11 5 end
end 1 q1
q1 N N1
q2 N N2
q3 N N3
q4 N N4
q5 N N5
q6 N N6
q7 N N7
q8 N N8
q9 N N9
q10 N N10
q11 N N11
N1 N N1
N2 N N2
N3 N N3
N4 N N4
N5 N N5
N6 N N6
N7 N N7
N8 N N8
N9 N N9
N10 N N10
N11 N N11
N1 1 q2
N2 1 q3
N3 1 q4
N4 4 q5
N5 4 q6
N6 1 q7
```

```
N7 1 q8
N8 5 q9
N9 4 q10
N10 1 q11
N11 5 end
```

5. Testing

Please find the code for testing my language below. It's written in Python and should compile with any python compiler of your choice.

You can also execute this code with an online compiler such as online-python.com, by copying and pasting the code below and clicking the run button on the website.

Alternatively, if you have python on your computer, you can download the same code in a .py file on my GitHub through this link and compile and execute it locally.

Nevertheless, I strongly recommend accessing the code via the GitHub repository versus copying and pasting from this pdf file, as I fear the page breaks will break the code or change the formatting. Since Python is rather particular about tabs, this would result in errors during execution.

I have already loaded the data structure from part 4 for your convenience. All you have to do is add calls to the accept function at the bottom of the py file. I have left some examples of pass and reject calls.

Python Code:

```
def read_automaton_from_string(data_str):
    lines = data_str.strip().split('\n')
    automaton = {
        'states': set(lines[0].split()),
        'alphabet': set(lines[1].split()),
        'start_state': lines[2].strip(),
        'accept_states': set(lines[3].split()),
        'transitions': {}
    }

    for line in lines[4:]:
        parts = line.strip().split()
        if len(parts) == 3:
            from_state, symbol, to_state = parts
            if from_state not in automaton['transitions']:
                automaton['transitions'][from_state] = {}
            if symbol not in automaton['transitions'][from_state]:
                automaton['transitions'][from_state][symbol] = []
            automaton['transitions'][from_state][symbol].append(to_state)

    return automaton
```

```
def accept(A, input_string):
    def helper_cleaner(s):
        note_chars = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'A', 'B', 'C', 'D',
             'E', 'F', 'G'}
        return ''.join(['N' if char in note_chars else char for char in s])

    cleaned_string = helper_cleaner(input_string)
    symbols = list(cleaned_string)

    current_states = [(A['start_state'], [A['start_state']])]

    for symbol in symbols:
        next_states = []
        for state, path in current_states:
            if state in A['transitions'] and symbol in A['transitions'][state]:
                for next_state in A['transitions'][state][symbol]:
                    next_states.append((next_state, path + [next_state]))

        if not next_states:
            return 'reject'
        current_states = next_states

    for state, path in current_states:
        if state in A['accept_states']:
            return ('accept', path)

    return 'reject'

twelve_bar_NFA_automaton_data = """
q0 q1 q2 q3 q4 q5 q6 q7 q8 q9 q10 q11 N1 N2 N3 N4 N5 N6 N7 N8 N9 N10 N11 end
1 4 5 N
q0
end
q0 1 q1
q1 1 q2
q2 1 q3
q3 1 q4
q4 4 q5
q5 4 q6
q6 1 q7
q7 1 q8
q8 5 q9
q9 4 q10
q10 1 q11
q11 5 end
end 1 q1
q1 N N1
q2 N N2
q3 N N3
q4 N N4
q5 N N5
q6 N N6
q7 N N7
q8 N N8
q9 N N9
q10 N N10
q11 N N11
```

```
N1 N N1
N2 N N2
N3 N N3
N4 N N4
N5 N N5
N6 N N6
N7 N N7
N8 N N8
N9 N N9
N10 N N10
N11 N N11
N1 1 q2
N2 1 q3
N3 1 q4
N4 4 q5
N5 4 q6
N6 1 q7
N7 1 q8
N8 5 q9
N9 4 q10
N10 1 q11
N11 5 end
"""

A = read_automaton_from_string(twelve_bar_NFA_automaton_data)

print(accept(A, "111144115415")) # Most Basic 12 Bar Blues We Can Have
print(accept(A, "111N144N11N54N15")) # 12 Bar Blues With some melody

print(accept(A, "1abfd11GA1C44N1ab1N5N4N1N5")) # More complex 12 Bar Blues

print(accept(A, "a b c 1 4 5 A C D")) # Should fail
print(accept(A, "hello")) # Should fail
```

Expected Runtime Results With No Changes

```
('accept', ['q0', 'q1', 'q2', 'q3', 'q4', 'q5', 'q6', 'q7', 'q8', 'q9', 'q10',
     'q11', 'end'])
('accept', ['q0', 'q1', 'q2', 'q3', 'N3', 'q4', 'q5', 'q6', 'N6', 'q7', 'q8',
     'N8', 'q9', 'q10', 'N10', 'q11', 'end'])
('accept', ['q0', 'q1', 'N1', 'N1', 'N1', 'N1', 'q2', 'q3', 'N3', 'N3', 'q4',
     'N4', 'q5', 'q6', 'N6', 'q7', 'N7', 'N7', 'q8', 'N8', 'q9', 'N9', 'q10',
     'N10', 'q11', 'N11', 'end'])
reject
reject


** Process exited - Return Code: 0 **
```