Hacking Kakofonix

Tomek Garbus

August 2024

1 Introduction – about Kakofonix

Before I say anything about the app, you can give it a try at https://tomaszgarbus.github.io/kakofoni/.

Kakofonix is my lightweight project for generating ~~music~~ cacophony. It was supposed to be just a warm-up to get familiar with Vue and ToneJS before moving on to a bigger app, but it stuck with me longer than expected.

The whole idea behind Kakofonix is to input mathematical sequences – of natural numbers modulo 13 to be specific – and playback them on virtual piano. Let's unwrap this sentence:

Defining sequences

Sequences in Kakofonix are defined as step transforms. User can create a set of variables $x_1, x_2, ..., x_n$ and assign initial values at step 0: $x_{1,0}, x_{2,0}, ..., x_{n,0}$. Then, the value for variable x_k at step s is defined as a step transform using arithmetic operations $+$, $-$, $\times$, $\hat{}$ (power), constants and evaluations at previous step: $x_{i,s-1}$.

Playback

Not all sequences have to be played, some can be used "silently" to define other sequences. For each played sequence, you can choose in which octave it will be played. For instance, if a sequence is played in the 2nd octave, its values at consecutive steps map to sounds: $0 \to C2, 1 \to C\#2, ..., 11 \to B2, 12 \to C3$.

There is currently no way to input rhythm, everything is played as straight eight notes.

Why modulo 13?

Western music system typically uses 12 notes, so why use arithmetics with modulus 13?

The main reason is that 13 is prime so it makes room for more interesting and balanced sequences. In addition, this allows us to incorporate an octave jump in the melodies.

Hands on!

If you haven't yet, head over to https://tomaszgarbus.github.io/kakofoni/ and try playing different presets or create your own.

2 Problem statement

After developing the app, I played a little bit with listening to random sequences, but then I wondered: how can I input something more structured, for example is it possible to encode 12 bar blues? Shortly later my friend started nagging me to add more features (e.g. arithmetic operators, fractional numbers) to the app because he wasn't able to build out a nice melody.

So the question was: do we really need to extend the app's semantics, or should the user get more clever with the math?

Or to be more precise: given an arbitrary melody $\mathcal{M} = m_0, m_1, ..., m_l$, where $0 \leq m_i < 13$, how to create a Kakofonix config such that one of the variables will play back $\mathcal{M}$ for the first l steps?

3 How to encode arbitrary ~~sequence~~ melody

of length 13

Let's start with sequences of length 13 or shorter: $m_0, m_1, ..., m_{12}$. There exists a unique polynomial $f(x)$ such that:

$$\begin{aligned} f(0) &\equiv m_0 \mod 13 \\ f(1) &\equiv m_1 \mod 13 \\ &... \\ f(12) &\equiv m_{12} \mod 13 \end{aligned}$$

It's pretty easy to find using standard polynomial interpolation techniques, such as Newton or Lagrange interpolation. They generalize to modular arithmetics by using modular division.

Now, how do we encode it as a Kakofonix config? We'll need a "counter" variable `0` with initial value 0 and step transform `x := x + 1`. The melody then can be encoded with variable `y` with initial value m_0 and step transform `y := a`$_0$ `+ a`$_1$ `* x + a`$_2$ `* x^2 + ... + a`$_{12}$ `* x^12`, where a_i are coefficients of the interpolated polynomial.

That's a nice finding, but a melody of 13 8th notes won't be very compelling. Can we do better?

of length 169

It's easy to see why this method above doesn't scale to higher degree polynomials, and therefore to longer melodies. What if instead of a polynomial we tried to interpolate some other function, this time with two arguments?

Let's use two helper sequences, x_i and y_i, such that $x_i = (i \mod 13)$ and $y_i = (\lfloor \frac{i}{13} \rfloor \mod 13)$. It's easy to see that the sequence of pairs (x_i, y_i) loops every 169 steps.

To encode y in Kakofonix config, we create a variable `y` with initial value 0 and step transform `y := y + y_delta`, where `y_delta` encodes melody $\underbrace{0, 0, ..., 0}_{11}, 1, 0$.

Consider the sequence of tuples (x_i, y_i, m_i), which expands to:

$$\begin{gathered} (0, 0, m_0), (1, 0, m_1), ..., (12, 0, m_1), \\ (0, 1, m_{13}), (1, 1, m_{14}), ..., (12, 1, m_{25}), \\ ..., \\ (0, 12, m_{156}), (1, 12, m_{157}), ..., (12, 12, m_{168}) \end{gathered}$$

If we an interpolate this sequence with a two-argument function, we'll be able to encode $\mathcal{M}$ as Kakofonix preset. Trying to generalise our polynomial-based solution from the previous section, let's see how far we can get with a function:

$$f(x, y) = \sum_{0 \le i,j < 13} w_{i,j} x^i y^j$$

where we need to figure out what are $w_{i,j}$. Let's write it down as a system of equations:

$$\begin{aligned} (x = 0, y = 0) \quad & f(x) = \textstyle\sum_{0 \le i,j < 13} w_{i,j} x^i y^j = m_0 \\ (x = 1, y = 0) \quad & f(x) = \textstyle\sum_{0 \le i,j < 13} w_{i,j} x^i y^j = m_1 \\ ... \quad & ... \\ (x = 12, y = 0) \quad & f(x) = \textstyle\sum_{0 \le i,j < 13} w_{i,j} x^i y^j = m_{12} \\ (x = 0, y = 1) \quad & f(x) = \textstyle\sum_{0 \le i,j < 13} w_{i,j} x^i y^j = m_{13} \\ ... \quad & ... \\ (x = 12, y = 12) \quad & f(x) = \textstyle\sum_{0 \le i,j < 13} w_{i,j} x^i y^j = m_{168} \end{aligned}$$

We have 169 equations with 169 variables (variables being $w_{i,j}$). The final step is to solve it with Gaussian elimination and encode $f(x, y)$ as a Kakofonix preset. Note that we could have also used Gaussian elimination for interpolating the polynomial in the previous section.

of any length

Hopefully, by now it should be clear how to generalize this method to sequences of any length that's power of 13. For a melody of length 13^k, we'll need k variables, where j-th variable takes values $\underbrace{0, 0, ...}_{13^j}, \underbrace{1, 1, ...}_{13^j}, \underbrace{2, 2, ...}_{13^j}$ and interpolate a function with k arguments.

4 Exercise for the reader

Prove that every melody within one octave is representable as Kakofonix config, that is the system of equations presented above always has a solution.

5 Implementation

The implementation is available at https://github.com/tomaszgarbus/mtlk/blob/main/kakofoni_hacks/main.py.