Aleatora II: Interfaces

Ian Clester — MUSI 7100

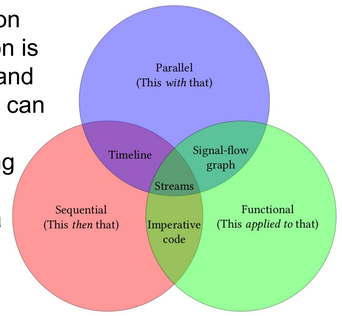
INTRODUCTION

Aleatora is a music programming framework built on streams that eschews divisions between synthesis and control. This semester, I have expanded outward from the core Aleatora framework, working on the other components needed for a more complete compositional environment. These are all interfaces: a **graphical interface** for managing compositional resources, an **instrumental interface** for working with external musical data (audio + MIDI), and a **networking interface** for composing with/for external streams of data in real-time. Together, these interfaces give the composer powerful new tools for weaving interactive, aleatoric, or externally-influenced elements into structured compositions.

RECAP: STREAMS

In Aleatora, **compositions are built out of streams** — chains of computation that generate a sequence of values**.** Aleatora's streams are lazy (computation is deferred until a value is required) and effectful (they may have side-effects, and these will occur again on re-evaluation), and they may or may not end. They can also be replayed from an earlier point, with a potentially different result.

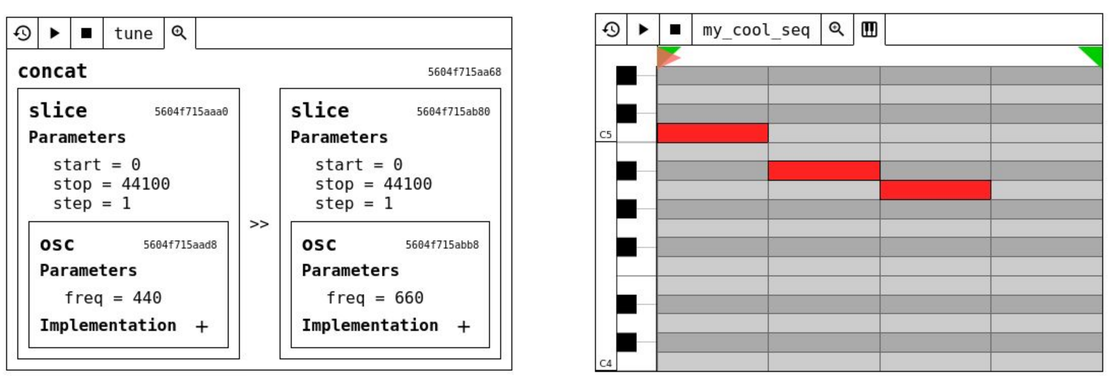
Aleatora's streams support **sequential composition** (one stream following another; this requires the concept of endings), **parallel composition** (two streams playing at the same time), and **functional composition** (applying a function to a stream to transform it). In this way, you can build a musical composition out of smaller, simpler pieces, in whatever ways make sense.



INTERFACE: GRAPHIC

Aleatora is a library for the Python programming language, so you create music with Aleatora by writing code. However, as DAWs and graphical languages like Max and Pd have demonstrated, it is valuable to have **an interface beyond text**: a rich interface for managing and manipulating musical resources, with support for trying things out live and recording new ideas.

In analogy to the DAW, which makes it easy to work with digital data, I have been working on a graphical assistant which might be described as a "CAW", a **computational audio workstation** for working with "computational data" — data that may not be known in advance, or may change each time it is used. This tool makes it easy to **play** different streams, **inspect** streams to see their structure, **run** code and hear the result, and **manage** many kinds of musical resources — including some that are easier to deal with visually than textually, such as **envelopes** and **sequences** (piano rolls).



Excerpts from the Aleatora assistant interface: stream inspection (top left), sequencer stream (top right), Python console/REPL (bottom)

INTERFACE: INSTRUMENT

It is possible to make music entirely "in the box" with Aleatora (solely from code), but **working with external material** is often essential. Last semester, Aleatora supported loading audio files into streams, and now it supports **live audio input**, which may be **composed like any other stream**. `foo >> input_stream[:20.0] >> bar` will play `foo`, then live input for 20 seconds, then `bar`; `input_stream + osc(40)` will play live input forever, with a drone. Since `input_stream` is real-time (and cannot know the future nor an unlimited past), it relies on the "side effect" of time passing externally: asking for any value in the stream always returns the latest sample. Input can be preserved through freezing or memoization, as with any other stream.

Aleatora now **supports MIDI** input via the Mido library. `event_stream` yields events from an input port, while `file_stream` yields events from a file. Both of these work with the **instrument interface** in Aleatora, which transforms a stream of messages interspersed with `None` into a stream of samples. This interface is especially supported by the assistant, which allows playing instruments live and recording performances. The function `render` writes out an event stream as a MIDI file.

Among other synthesized instruments, Aleatora supports text-to-speech via Google or the open-source Festival library, and it supports a subset of FoxDot's pattern notation and play strings.

INTERFACE: NETWORK

Aleatora's networking module exposes **network streams** as Aleatoric streams and facilitates their integration into compositions. `byte_stream` exposes a **TCP connection** as a stream of bytes, `packet_stream` returns a stream of **UDP datagrams**, and `osc_stream` returns a stream of **OSC messages**. At a higher level, `request_stream` issues **repeated requests** over HTTP and yields the responses, and it is straightforward to **streamify third-party libraries** (such as those for specific APIs) via `repeat`, which returns a stream that repeatedly calls a given function. `unblock` converts a potentially blocking stream into a non-blocking stream (returning a filler value or holding the last value until the next is ready) and may be used in conjunction with any of the network streams.

DISCUSSION AND NEXT STEPS

My immediate goal is to prepare Aleatora for release. Over the summer, I plan to refine and document the framework, improve performance, and investigate the possibility of running in the browser. For the assistant, I aim to make it more complete and usable, and investigate the possibility of building off infrastructure from the Jupyter project (so as to avoid reinventing interactive-computing wheels).

Looking further ahead, Aleatora is the only first in a three-part project on computational compositions. There are two main areas of focus for the future: **distribution** and **accessibility**. For distribution, the question is: **How can computational compositions be distributed so that they are easy to share and play?** (For example, think of how easy it is to share and play recordings on Spotify or SoundCloud.) For accessibility, the question is: **How can musicians who do not code benefit from the possibilities of computational compositions?**

FURTHER READING

For more on Aleatora, you can read this semester's paper at (ijc8.me/aleatora-ii-preprint.pdf), last semester's paper at ijc8.me/aleatora-i.pdf, and the Aleatora Guthman Fair video at youtu.be/hcC3Mhp6PdE. This poster is available at ijc8.me/aleatora-ii-poster.pdf, and last semester's is available at ijc8.me/aleatora-i-poster.pdf