1 Introduction

The great proliferation of computer music programming languages points to the difficulty of building a natural interface for users who want to computationally interact with musical data. Programming applications in the domain of computer music, and specifically digital signal processing (DSP), requires that users not only grasp fundamental programming techniques, but also have a large domain specific knowledge on time and signal manipulations. The amount of prerequisite skill and effort to overcome these barriers is often higher than many users are able to commit.

Furthermore, the difficulty of programming DSP applications is often not commensurate with the creative intentions. As a motivating example, imagine a user was to reconstruct the filter that was used to transform an audio clip. In this case the user has the original audio file, and the transformed audio file, but does not know how this transformation happened. In the standard approach, a user would need to be a domain expert and listen to the two files, and aurally estimate which kinds of filters were used to achieve the transformation. Once the user has some suspicion as to the appropriate filter types that will be needed, the user must write a program in some language (SuperCollider, MaxMSP, CSound, etc) that implements the DSP filter the user has in mind. Further still, the user will then need to spend time tweaking the filter parameters to find the best fit.

To simplify this process, we use DSP programming by example (DSP-PBE). With DSP-PBE, the user simply provides our tool with the original audio, and the transformed audio, and will automatically receive a DSP filter that approximates the transformation. This was the approach used for the audio files in Figure 1. In this example, a user provided a clip of a ‘cartoon spring’ in Figure 1a, and the same sound as it had been transformed with a low-pass filter at 800 Hz, \( \text{lp f (800)} \), as shown in Figure 1b. However the nature of the transformation is unknown to the user and they wish to discover the filter needed. Our DSP-PBE tool is able to synthesize a filter \( \text{lp f (947)} \), that when applied to the original sound, produces the waveform shown in Figure 1c. While
the solution is not exact, the difference is not significantly noticeable to an untrained ear.

The problem of DSP programming by example is formally defined as follows: Given an input waveform $I$ and an output waveform $O$, construct a DSP filter $F$, to minimize the aural distance $\text{dist}$ between the $O$ and $F(I)$. In a single line:

$$\text{Find } F, \text{ such that } \text{dist}(O, F(I)) = 0$$

The two key components of this statement are the definition of distance ($\text{dist}$) and a search technique to find $F$. A distance metric that is faithful to the psycho-acoustics of the human ear is critical for a useful tool. As an example, taking a trivial distance function that returns the difference in length of the two audio samples will allow a delay filter to satisfy any example pair of samples.

Additionally, an efficient search algorithm is critical, as the space of possible DSP filters is very large. Not only do we need to consider a wide variety of filters, we need to consider the space of parameters for each filter, as well as the different ways of combining multiple filters.

Currently, Ruzica Piskac and her team have a DSP-PBE Synthesizer that works on commutative sound samples. The distance calculation they used will remain constant for the terms of my project. My task is to figure out how to apply their work to non-commutative sound samples by altering/designing the search algorithm used.

## 2 Search Algorithm

Search techniques for programming by example have been the subject of intense research. Because the search space of possible programs is extremely large, search procedures must be exceptionally efficient.

Gradient descent could be a viable search technique. It is the current search process used by Piskac’s team. However, one trouble with this approach is that we have lots of local minimums in the space of optimization. Let’s take the example of trying to learn a low-pass filter threshold. If our initial threshold is too low (cutting off too much), and just on the other side of the frequency we need, SGD will easily send us in that direction. However, if the spectrogram of waveform has a frequency spectrum that is mostly inactive, SGD will detect a
plateau and tell us we have found a global minimum, when in fact we are stuck at a local minimum.

So, in order to overcome this problem, we can add additional techniques to the search algorithm to help us find a filter. Refinement types are a way of giving an abstract description of the behavior of a function. For example, we can describe the function $\text{map} :: [a] \to [b]$, that captures some properties of the behavior of the function, $f :: \text{xs} :: [a] \to \text{ys} :: [b] \mid \text{length xs} == \text{length ys}$, in this case that the length of the lists are still equal. In a similar style for DSP, we can write predicates about the filters available to us during synthesis. For example, a low-pass filter could be described as a refinement type that says the amplitude of the frequencies greater than the threshold have decreased in the output Audio. $\text{lpf} :: \text{threshold} :: \text{Float} \to \text{xs} :: \text{Audio} \to \text{ys} :: \text{Audio} \mid \text{amp(threshold, xs)} < \text{amp(threshold, ys)}$

One hypothesis to how we would combine both algorithms would be through a feedback loop. We will use the refinement types to select a structural part of an aural distance graph created by the $\text{dist}$ function above. We will then use gradient descent to find the local minimum in this structural part. If this minimum is below some threshold we created with our refinement types, we will accept the structural part we chose. If it is not, we will find a new structural part and the loop will continue. An example of this can be seen in the diagram below:

This current hypothesis for finding a filter $\mathcal{F}$ as mentioned above is where I will begin my research for this project.

### 3 Deliverables

The culmination of this project will result in the following deliverables:

1. Survey overview of related work in synthesis and DSP and how it can drive this project
2. Research and design a search algorithm for this project
3. Evaluation of algorithm focused on non-commutative filters