Signal Analysis and Music Processing Language Proposal

Mike Haskel, Mike Glass, Morgan Rhodes, and Navarun Jagatpal

February 7, 2007

SAMPL is a simple, functional, strictly typed, potentially platform-independent, translated, high-performance signal processing language.

1 Introduction

SAMPL is a language used for signal and music processing. It produces programs that take in audio streams as input and outputs the modified audio stream along with text output. SAMPL also allows for the sampling of the audio file to be transparent to the user, with the user defining desired behavior for the program and the actual loop for the sampling happening in the background. As a program designed for the processing of music, it has domain-specific types. SAMPL has primitive types including time, frequency, and intensity, along with string and integer types. SAMPL has the potential for platform-independence as it will be translated to C. Possible applications of SAMPL are as simple as frequency filters, effects(such as reverb), and as complex as identifying and filtering a particular instrument.

2 Functional

SAMPL is a functional programming language in which programs define the output stream in terms of various operations on the input stream. Programs consist of a series of recursive definitions. The language provides functions for sampling the input stream, such as

```
frequency(200 hz, 5 s)
rmsvolume(500 ms)
```
i.e. samples of the input which vary over time
and
```
clip(stream, 50db)
mix(stream1, stream2)
highpass(stream, 400hz)
```
i.e. functions for mutating streams to produce other streams

3 Strictly Typed

SAMPL is strictly typed, providing types directly related to signal processing such as frequency, time, and intensity. It also provides string and integer types. Programs specify values of these types by specifying units, as in 300 hz, 20 db, or 50.3 ms. The language supports basic operations such as comparing, adding, or scaling values of a given type, and provides functions for converting between types in meaningful ways (such as frequency–time).

4 Control Flow

One of the key features of SAMPL is that the process of scanning and sampling input is transparent to the user. Ordinarily, a program for signal processing would need to implement a tight loop which encapsulates the process of reading the input and updating state as necessary. This process is conceptually distinct from the specification of the audio transformations and users shouldn't need to implement it. To accommodate this, SAMPL implements this loop in the background, and a program's control flow is based upon constructs for applying filters when certain conditions hold.

Such constructs include "if-then-else", which takes on a different value depending on some condition, as in

```
if rmsvolume(500 ms) > 60 db
   lowpass(input, 5000 hz)
else
   input
end
```

which applies a low-pass filter to the input when and only when the input is sufficiently loud. Another important construct is "wait-until", which transitions between values when specified conditions are met, as in

```
flute_playing =
  wait
    false
  until flute_attack
    true
  until flute_dropoff
    flute_playing
  end
```

which defines flute_playing to be false until a flute starts to play, then true until it stops, then repeating as if from the beginning. These control structures limit the layers of indirection between the programs and their intended effects.

5 Possible Applications

SAMPL has a number of possible applications, ranging from simple to complex processing tasks. Simple applications in SAMPL will allow low or high pass filters to be applied to the stream, outputting the filtered stream. Another application will be applying reverb to the input stream. These applications can be made more complex in applying a high pass filter whenever a there is a frequency lower than a specified frequency and apply reverb only when the intensity exceeds a certain value. SAMPL will also make it possible to write applications to recognize particular instruments which will make it possible to filter out the flute, or only filter out the flute while there is a trumpet playing. Programs generated by the SAMPL compiler are themselves inherently modular, as users can easily pipe output from one program to another.