Finding Mathematical Structures in Arts

Sounds and Tunes

Yusuke Kiriu

StudioPhones, Japan
uskusk@cwo.zaq.ne.jp

Kosaku Nagasaka

Kobe University, Japan
nagasaka@main.h.kobe−u.ac.jp

Tadashi Takahashi

Kobe University, Japan
takahasi@kobe−u.ac.jp

Recently, computer aided composition and adaptation and computer music are quite popular. For example, in the last International *Mathematica* Symposium, WolframTones was introduced, which is a kind of computer aided composition and adaptation. Moreover, by the recent rapid advance in computer science, everybody can compose and enjoy their sounds and tunes on their personal computers. Professional musicians have not been only the person who can create music anymore. New paradigm must be coming in Art (not only in computer arts). We present some packages and current results from this point of view.

■ Introduction

The aim of our project is finding mathematical structures in Arts. We are not interested in computer aided composition and adaptation by some mathematical theories: cellular automata and Markov random field for example. Our purpose is another direction: "Mathematics on Arts", or we'd like to find some universal mathematical structures in Arts. At this time, we have not yet found any universal structure in Arts. However, we have developed and been developing Ysk and Ksk sound packages with a new concept: "structure oriented sound object" like object oriented programming. By those packages, we have been finding some universal structures in Arts.

Ysk package is mainly for re−factoring sound and tunes schemes to be compatible with mathematical knowledge. The most significant functionality of this package is that we can operate (connect, combine, shift, making harmonic sounds and so on) with sound objects by symbolic formulations like basic four operations for numbers. Implementing this package has been almost finished. Ksk package is mainly for re−capturing sound and tunes schemes as mathematical objects though it is still under construction.

By those package, we will study the following three things: 1) re−factoring sound and tunes schemes with mathematical knowledge, 2) re−capturing sound and tunes as mathematical objects, and 3) re−factoring sound and tunes as mathematical objects. Moreover, at the presentation, we discuss about some notes on *Mathematica*'s sound system for professional use.

Moreover, gaining factoring sounds and producing sounds is two side of the same coin. Mathematics is the science of "explanation" (reducing everything to "plain things"). By using computer algebra system, we can reduce the technics of complicated sounds to simple steps. The development of Ysk and Ksk is an exciting study of mathematics and sounds (music). There are many systems for analyzing sounds (music). We will show the approach using computer algebra system "*Mathematica*". The ultimate goal of technology and sounds is mathematics. Ysk and Ksk is a powerful tool for researching sounds. Music/Mathematics researchers should appreciate the possibility of sharing cognitive level with such technology. Music/Mathematics researchers will be able to use Ysk and Ksk as a 'partner' in the future.

■ Ysk Package for Re−factoring

In this section, we show some results from our project in terms of Ysk package whose main purpose is for re−factoring sound and tunes schemes with mathematical knowledge. The keyword is "Sound and Tunes can be operated by mathematical expressions". Ysk package gives a sound create environment on which we can make sounds and tunes by comprehensive symbolic computations. By the package, we can follow most of functionalities of various kind of sound researching softwares in the world as symbolic computations.

This loads the package.

In[1]:=
```
<< Ysk`YskSound`
```

```
implemented by Yusuke Kiriu (c) 2005.
```

We can play 220Hz sine wave with an appropriate envelope function.

In[2]:=
```
a1 = Play[Sin[220 * 2 * Pi * t] * (1 - Tanh[3 * t]),
    {t, 0, 1}, PlayRange → {-1, 1}];
```

You could not hear any sounds if you read this on a printed material. If so, we encourage you to read with your imagination that all mathematical expressions are represented as sounds.

Here, in Ysk package, we suppose that sounds are formed by a list of amplitude as in the conventional sound devices, though there are other definitions for sounds. Since a list of lists of amplitude must be the list of amplitude, this symbolic operator corresponding to join sound objects plays the sound which are formed by these three sound objects.

In[3]:=
```
□{a1, a1, a1}
```

Out[3]= - Sound -

Shifting frequencies can be done by a symbolic way.

In[4]:= □ {a1, a1, 1.2⊖a1}

Out[4]= - Sound -

Making harmonies also can be done by the additive symbol "+". This plays a sound and the harmony of this and the previous sounds.

In[5]:=
```
a2 = Play[Sin[250 * 2 * Pi * t] * (1 - Tanh[3 * t]),
    {t, 0, 1}, PlayRange → {-1, 1}];
```

In[6]:= □ {a1 + a2, a1, a2}

Out[6]= - Sound -

By these symbolic operators which are well–known as basic four operations, we can do conventional sound operations.

In[7]:= □ {a1, a1 - a2, 1.2⊖a1}

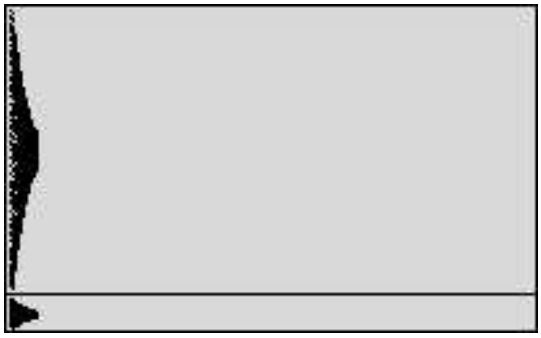
Out[7]= - Sound -

Moreover, these operators can be elements of complex algorithms making sound and tunes. A Rhythm is constructible as a list of frequency and also can be generated by the following symbolic way.

```
In[8]:= tab1 = Table[ 2^(i / 12) ⊖ a1, {i, 0, 12}]

Out[8]= {- Sound -, - Sound -, - Sound -, - Sound -,
         - Sound -, - Sound -, - Sound -, - Sound -,
         - Sound -, - Sound -, - Sound -, - Sound -, - Sound -}

In[9]:= □%
```



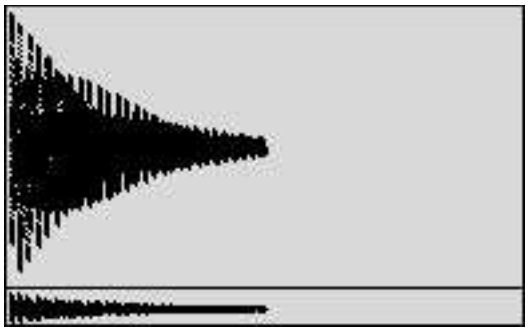
```
Out[9]= - Sound -
```

Since it is a symbolic operation, we can make a harmony by extracting sounds from the above rhythm list and overlapping them. This extracts the first, 5−th and 8−th sounds from the above list and makes a harmony.

```
In[10]:= Plus @@ tab1[[{1, 5, 8}]]

Out[10]= - Sound -

In[11]:= □%
```



```
Out[11]= - Sound -
```

These symbolic operations enable us to construct complex sound structures with explicit definitions and also enable us to visualize the structure, definition and approach of the sound object generated, easily.

This defines a sound track.

```
In[12]:= tr1 = UskSoundTrack[{a1, a1, 1.2⊖a1}]

Out[12]= - SoundTrack -
```

We can do some list operations for the sound track defined above.

```
In[13]:= tr2 = UskSoundTrackRotateLeft[tr1, 2]

         - SoundTrack -
```

After evaluating the sound track, it becomes a normal sound object. Before evaluating, it keeps its own symbolic structure hence we can do other symbolic operations for it.

In[14]:=
```
▽tr2
```

Out[14]=
```
- Sound -
```

Other noteworthy structure is the following sound array.

In[15]:=
```
UskSoundArray[{{a1, a1, 1.2⊖a1}, {a2, a2}}]
```

```
- SoundArray -
```

Even if they are sound arrays, we can operate them by symbolic ways.

In[16]:=
```
tr1 + tr2
```

```
- SoundArray -
```

Since this package is not for real−time sound playing, we can operate with complex and time−consuming waves. Moreover, we can operate by *Mathematica*'s rule based programing. We can play sounds by specifying lightweight waves via GUI as if it is a sound device to play.

```
SimpleSinWaveGenerator[]
```

We note the following point. Most of so−called pianos or instruments are optimized for players by historical human studies. For this common sense, we have a question: is it required for mathematical analyses or as mathematical objects? We think that we can reflect mathematical properties or structures to GUI based instruments. For example, the well−known such instrument is Theremin.

Representing instruments in terms of mathematical science is our interest. The above `SimpleSinWaveGenerator` example is an instrument which plays sounds generated by slider operations for frequency. We also think that we can not neglect the conventional piano keyboard for conventional uses. Here, we demonstrate such a piano keyboard by using a computer keyboard. Unfortunately, on a printed version of this document, we can not show the result since this program plays sounds corresponding keys down (eg. D, R, F, T, G, H, U, J, I, K, O and L for 12−note scale).

```
Module[{major, minor, a, b, c},
 major = {0, 2, 4, 5, 7, 9, 11};
 minor = {5, 7, 9, 11, 12, 14, 16};
 a = Table[Mod[5 * n, 12], {n, 0, 7}];
 ko[mm_] := a[[mm]] + major;
 b = Map[ko, Range[8]];
 kp[mm_] := a[[mm]] + minor - 12;
 c = Map[kp, Range[8]];
 Join[{b, c}]]
```

```
{{{0, 2, 4, 5, 7, 9, 11}, {5, 7, 9, 10, 12, 14, 16},
  {10, 12, 14, 15, 17, 19, 21}, {3, 5, 7, 8, 10, 12, 14},
  {8, 10, 12, 13, 15, 17, 19}, {1, 3, 5, 6, 8, 10, 12},
  {6, 8, 10, 11, 13, 15, 17}, {11, 13, 15, 16, 18, 20, 22}},
 {{-7, -5, -3, -1, 0, 2, 4}, {-2, 0, 2, 4, 5, 7, 9},
  {3, 5, 7, 9, 10, 12, 14}, {-4, -2, 0, 2, 3, 5, 7},
  {1, 3, 5, 7, 8, 10, 12}, {-6, -4, -2, 0, 1, 3, 5},
  {-1, 1, 3, 5, 6, 8, 10}, {4, 6, 8, 10, 11, 13, 15}}}
```

```
PianoKeyboardDemo[]
```

As in the above examples, by this package, we can combine several sound operations implemented for *Mathematica* and conventional forms of playing sound and tunes. Moreover, there are special three structures, `SoundFormSounds`, `SoundFormList` and `SoundFormFunction` by which we can preserve several operations unevaluated, as follows. They are useful for building a database, operating with more abstractness and so on.

```
flo[freq_] := Play[Sin[freq * 2 * Pi * t] * (1 - Tanh[3 * t]) / 2,
  {t, 0, 1}, PlayRange → {-1, 1}, DisplayFunction → Identity]
```

```
a = Flatten[{flo /@ {220, 226}, .1, 2,
   1 / 3, 1.2⊖# &, UskSoundNullAddF[#, 44100] &,
   UskSoundNullAddR[#, 44100] &}]
```

```
{- Sound -, - Sound -, 0.1, 2, 1/3, 1.2⊖#1 &,
 UskSoundNullAddF[#1, 44100] &, UskSoundNullAddR[#1, 44100] &}
```

```
a2 = UskSoundForm[a]
```

```
- SoundForm -
```

```
a2[[1]]
```

```
{- SoundFormSounds -, - SoundFormList -, - SoundFormFunction -}
```

By using UskSoundFormFunctionTranspose, we can transpose only effects on each sound element of the sound form, which is similar to the latter example of Ksk package, though the approach and the context used are different from Ksk.

```
UskSoundFormFunctionTranspose[{a2, a2}]
```

```
{- SoundForm -, - SoundForm -}
```

```
#[[1]] & /@ %
```

```
{{- SoundFormSounds -, - SoundFormList -,
  - SoundFormFunction -}, {- SoundFormSounds -,
  - SoundFormList -, - SoundFormFunction -}}
```

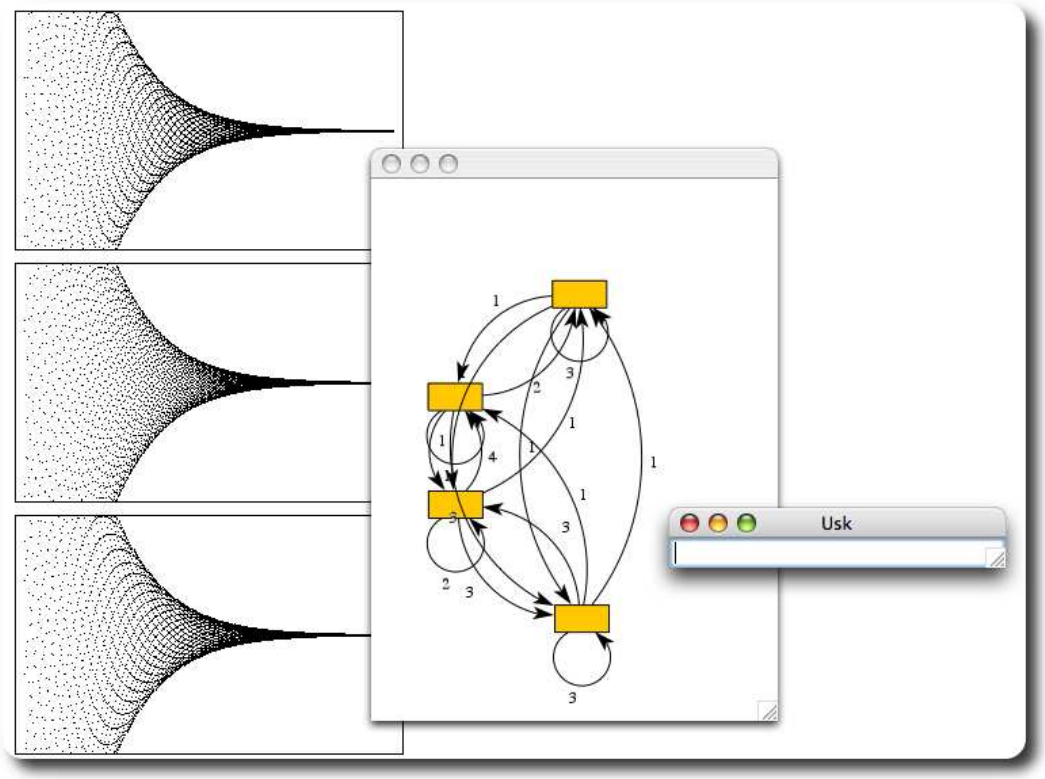
```
Map[#[[1]] &, #] & /@ %
```

```
{{{- Sound -, - Sound -}, {0.1, 2, 1/3}, {1.2⊖#1 &, 1.2⊖#1 &}},
 {{- Sound -, - Sound -}, {0.1, 2, 1/3},
  {UskSoundNullAddF[#1, 44100] &,
   UskSoundNullAddF[#1, 44100] &}}}
```

Another example is like GUIKit: Examples, GraphEditor. This plays sounds by using a computer keyboard with tone operations in each bar.

```
WithGraphEditorDemo[]
```

From In[2]:=



In this example, using GraphEditor, we modify the tone adjusting matrix for the unit bar.

We note that Ysk package has been implemented not only for sound and tunes but also other arts which can be represented by Graphics and Graphics3D.

■ Ksk Package for Re−capturing

Ksk package is for re−capturing sound and tunes as mathematical objects. Conventional operations for sound objects are usually result−oriented. We would connect sounds, combine sounds, shift a sound or make harmonic sounds for getting better sound outputs. There may not any known mathematical structures unfortunately. Our aim is to bridge a gap between the conventional operations and any known mathematical structures. By Ksk package, we can construct sound matrices and transpose them for example. Although the package is still under construction and is not fully functioning, some of functions have been implemented so we show them in this section. The picture is worth a thousand words and seeing is believing, hence we show some examples at first.

We note that our current result is tiny and a small step, it might be strange or an extreme argument from the artistic point of view since this is just an interim report.

□ Examples: What we can do by Ksk package

This reads the package.

In[1]:=
```
Needs["KskSound`"]
```

```
This KskSound package includes routines which provide
   extended sound functionalities for Sound objects
   on Windows machines. Other platforms are not
   supported yet. The version of this package is 0.0.1
  and implemented by Kosaku NAGASAKA (c) 2005-2006.
```

The package has the following functions.

In[6]:=
```
?KskSound`*
```

KskSound`

ClearAllGeneratedSound	SoundArray
ClearAllGeneratedSoundExceptFirstLevel	SoundArrayEffectFullTranspose
ClearGeneratedSound	SoundArrayEffectTranspose
ConnectThreshold	SoundArrayExpand
GainValue	SoundArrayFullExpand
GainVariableInForm	SoundArrayFullTranspose
GeneratedSound	SoundArrayTranspose
InterpolationRate	SoundConnect
KeepGeneratedSound	SoundCutOff
KskSound	SoundGain
KskSoundForm	SoundIdentity
LinearCombinationWithAverageLine	SoundMakeArrayed2D
LinearCombinationWithEdgeLine	SoundMakeFullArrayed2D
LinearCombinationWithSinInterpolation	SoundNormalize
Normal	SoundOverlap
ResultAsFunction	SoundOverlapAtFirstLevel
Scaling	SoundTuck
Show	

This generates a few *Mathematica*'s simple sound objects which will be elements of our structured sound objects.

In[3]:=
```
envelope = Function[(Log[#1 + 0.001] - Log[0.001]) E^(-8 #1) / 3]
```

Out[3]=
$$\frac{1}{3}\left(\text{Log}[\#1 + 0.001] - \text{Log}[0.001]\right) e^{-8\,\#1}\ \&$$

In[4]:=
```
s300e = Play[envelope[t] * Sin[2 * 300 Pi t], {t, 0, 1}];
```

In[5]:=
```
s400e = Play[envelope[t] * Sin[2 * 400 Pi t], {t, 0, 1}];
```

In[6]:=
```
s500e = Play[envelope[t] * Sin[2 * 500 Pi t], {t, 0, 1}];
```

In[7]:=
```
s600e = Play[envelope[t] * Sin[2 * 600 Pi t], {t, 0, 1}];
```

This adjusts the sound volume to its 50%. By KskSoundForm, we can check the structure of the output. To play the sound objects, as *Mathematica*'s plain sound objects, we can use Show in the KskSound context.

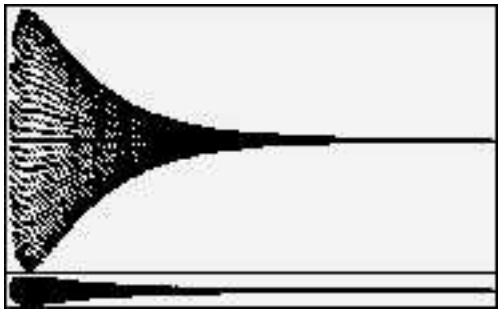
In[8]:=
```
sndA = 0.5 * s300e
```

Out[8]=
```
- Sound -
```

In[9]:=
```
KskSoundForm[sndA]
```

```
0.5 ( - Sound - )
```

In[10]:=
```
Show[sndA]
```



Out[10]= - Sound -

This makes simple structured sound objects by overlapping and connecting sounds.

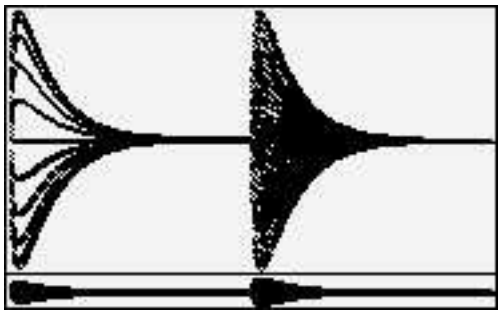
In[14]:=
```
sndB =
 SoundConnect[{0.75 * s400e, SoundOverlap[{sndA, s600e}]}]
```

Out[14]= - Sound -

In[15]:=
```
KskSoundForm[sndB]
```

```
( 0.75 ( - Sound - )   ( 0.5 ( - Sound - ) ) )
                         - Sound -
```

In[16]:=
```
Show[sndB]
```



Out[16]= - Sound -

For making structured sound objects, we can use SoundArray. This function helps us to make complex sound objects having matrix like structures.

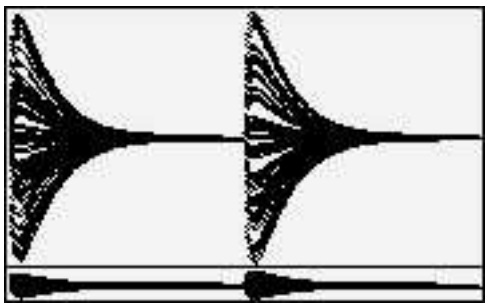
In[20]:=
```
sndC = SoundArray[
  {{0.3 * s300e, 0.4 * s400e}, {0.6 * s500e, 0.5 * s600e}}]
```

Out[20]= - Sound -

In[21]:=
```
KskSoundForm[sndC]
```

```
Array( 0.3 ( - Sound - )   0.4 ( - Sound - ) )
       0.6 ( - Sound - )   0.5 ( - Sound - )
```

In[22]:=
```
Show[sndC]
```



Out[22]=
```
- Sound -
```

This transposes the above matrix style structured sound object. In fact, in the output of KskSoundForm, we can see the difference between before and after transposing.

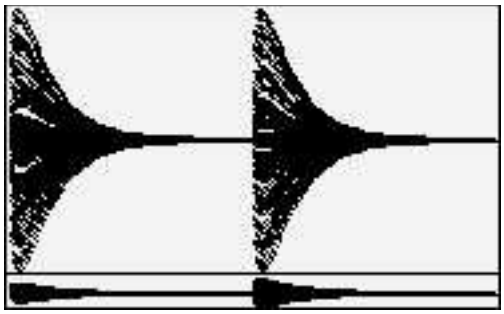
In[24]:=
```
sndD = SoundArrayTranspose[sndC]
```

Out[24]=
```
- Sound -
```

In[25]:=
```
KskSoundForm[sndD]
```

```
Array( 0.3 ( - Sound - )   0.6 ( - Sound - )
       0.4 ( - Sound - )   0.5 ( - Sound - ) )
```

In[26]:=
```
Show[sndD]
```



Out[26]=
```
- Sound -
```

This expands the matrix like structure into a combination of connecting and overlapping sound objects. Then, by `SoundMakeArrayed2D`, we build up a matrix like sound structure from the combination. These two sound objects are corresponding to the same sound though their representations are different. This relation is similar to the relation between $x^2 - 1$ and $(x + 1)(x - 1)$ for example.

In[28]:=
```
sndE = SoundArrayExpand[sndD]
```

Out[28]=
```
- Sound -
```

In[29]:=
```
KskSoundForm[sndE]
```

```
( ( 0.3 ( - Sound - ) )   ( 0.6 ( - Sound - ) ) )
  ( 0.4 ( - Sound - ) )     ( 0.5 ( - Sound - ) )
```

In[31]:=
```
sndF = SoundMakeArrayed2D[sndE]
```

Out[31]=
```
- Sound -
```

In[32]:=
```
KskSoundForm[sndF]
```

```
Array( 0.3 ( - Sound - )   0.6 ( - Sound - ) )
       0.4 ( - Sound - )   0.5 ( - Sound - ) )
```

In[33]:= **? SoundMakeArrayed2D**

```
SoundMakeArrayed2D generates a SoundArray object from
  SoundConnect and SoundOverlap objects. More...
```

Sound objects and effects for sounds are different elements or included in different domains, hence we can distinguish them like the following example. `SoundArrayEffectTranspose` transposes only sound effects. We note that multiplying a pure function adjusts the sound volume.

In[35]:=
```
sndG = SoundArray[{{0.5 * sndA, (4 # (1 - #) &) sndC},
    {0.5 * s400e, SoundNormalize[s500e]}}]
```

Out[35]= - Sound -

In[36]:=
```
KskSoundForm[sndG]
```

```
Array[
  0.5 ( 0.5 ( - Sound - ) )   4 (1 - t) t ( Array[ 0.3 ( - Sound - )   0.4 ( - Sound - )
                                                  0.6 ( - Sound - )   0.5 ( - Sound - ) ] ) ]
      0.5 ( - Sound - )                                [ - Sound - ]
```

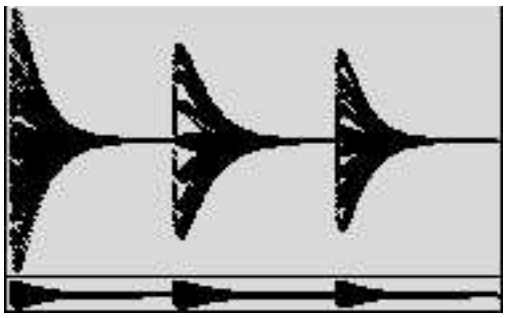
In[38]:=
```
sndH = SoundArrayEffectTranspose[sndG]
```

Out[38]= - Sound -

In[39]:=
```
KskSoundForm[sndH]
```

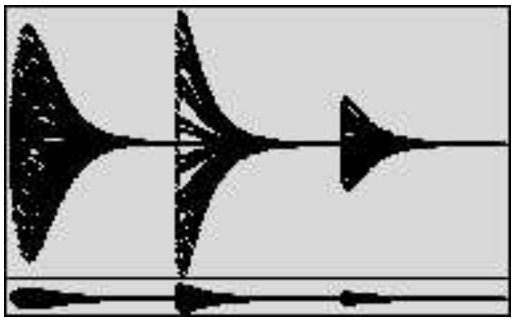
```
Array[ 0.5 ( 0.5 ( - Sound - ) )   0.5 ( Array[ 0.3 ( - Sound - )   0.4 ( - Sound - ) ] ) ]
       4 (1 - t) t ( - Sound - )                 0.6 ( - Sound - )   0.5 ( - Sound - )
                                                      [ - Sound - ]
```

In[40]:=
```
Show[sndG]
```



Out[40]= - Sound -

In[41]:=
```
Show[sndH]
```



Out[41]= - Sound -

In[42]:= **? SoundArrayEffectTranspose**

```
SoundArrayEffectTranspose transposes only the effects of
  the given square sound array or sound matrix. More ...
```

□ Re−capturing in terms of Ksk package

We mainly think of algebraic structures for re−capturing sound and tunes. What kind of algebraic structures are there on sound and tunes? Which is the best algebraic structure for them? We think that there may be so many possibilities while they are completely different histories.

Since the final aim of our project is searching for mathematical structures on sound and tunes, we would like to construct an algebraic structure like follows for example, and figure out several mathematical properties on the structure, which have some important meaning in terms of sound and tunes.

$$\mathbb{S} = \text{set of sound elements}, \quad \circ = \text{operation on } \mathbb{S},$$
$$\forall\, s, t \in \mathbb{S},\ s \circ t \in \mathbb{S}$$

At this time, the ℝ−module is one of candidates. Hence, in Ksk package, we have been trying to implement functions over vector spaces like matrix operations (eg. scalar multiplication, addition, transpose and so on). Moreover, for thinking of algebraic structures, we must think of that each element should be. This problem is related to a question: "What forms sound and tunes?". Does a combination of amplitudes form them? Does a combination of notes form them? We want to find an answer to this problem by our mathematical approach.

Currently, our Ksk package only can do as in the above examples. We will implement more functions like operations over some ℝ−modules.

■ Current Result and Remarks

Our project is formed by researchers for applied mathematics and sound and tunes. We have thought that the most important issue for cooperating with different subjects is "balancing". It might be "optimizing" if the purpose is very specific and specified. It may be an industrial way if our purpose is making some artistic works. It may also be by some mathematical models if the purpose is analyzing arts mathematically. As in the above, our strategy is different from these three ways, it is a two sidedness approach preserving mathematical and musical properties. Hence, we have been constructed two different packages Ysk and Ksk. We give some remarks from this point of view.

□ Using Ysk and Ksk Packages Together

Again, the aim of our project is finding mathematical structures in Arts. This subject has been studied by many people and some results are given. For example, Euler, Mersenne, Pythagoras and so on, those mathematicians lived some historical periods in terms of sound and tunes. There were some mathematical results: harmonics, rhythms and music theory, which may be not treated as important things in the history of music. Why? From our project, if we give the reason, we think that mathematical expressions are perfect and music is not and is changing since it is alive. Their main ideas are different so mathematics can not reach for music and vice versa.

We would like to get good responses for our result in terms of the both of mathematics and sound and tunes. One may think that Ysk and Ksk packages are the same on first sight, however, they are different since Ysk is made in terms of sound

and tunes and Ksk is made in terms of mathematics. By this approach, we think that we can get good responses from the both subjects.

For this purpose, we have been using *Mathematica*'s own help system to prepare enough documents about Ysk and Ksk. We have made effective use of these documents and realized differences between two subjects. This approach will help us to achieve our project. In fact, they have been very useful for communicating between different cultures, notations and people. Moreover, we think that they will be also useful tools for communicating in the future and we are looking forward results by which mathematics and sound and tunes meet together.

In fact, several functions must be used if we make a sound by Ysk or Ksk package. This means that the same functions or functionalities are required for each package to distinguish their approaches.

□ Future Work

We have studied so many things from our project. We realized that studying a subject in terms of other different subjects is quite meaningful. Though Ysk and Ksk packages have the same functionalities for operating *Mathematica*'s own sound objects by different ways, they are compatible each other in terms of Sound objects. Therefore, we can use several functions made from mathematical and musical point of view together. We think that blending two package leads us to the next step where we can study mathematical structures in sound and tunes. Blending two packages, making many models and observing their validities are our future works.