NEW SOUND SPACES FROM OLD: A BRIEF ON THE XI OPERATOR

CARL S. MCTAGUE

1. INTRODUCTION

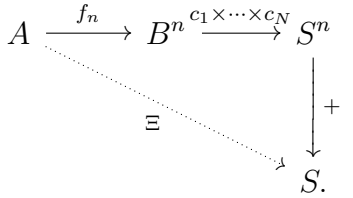
Let S denote the space of all possible sounds with sound superposition as addition. Suppose a space A is equipped with a mapping $A \xrightarrow{c} S$. Then we call the pair (A, c) a *sound representation space*, or *sound space* for short. The objective of this paper is to present a useful, general technique for inductively constructing new sound spaces from old.

2. THE Ξ-OPERATOR

Suppose $\{A \xrightarrow{f_n} B^n\}_{n\in\mathbb{N}}$ is a family of maps and $\{(B, B \xrightarrow{c_i} S)\}_{i=1}^{N}$ a finite sequence of existing sound spaces (all defined on the same space B). Then we can construct a natural sound space $(A, A \xrightarrow{\Xi} S)$ by setting

$$\Xi(a) := \sum_{i=1}^{N} c_i(f_N^i(a))$$

where $f_N^i(a)$ denotes the ith component of $f_N(a)$. Diagrammatically, Ξ is the dotted map in the commutative diagram:



We call the members of the families $\{A \xrightarrow{f_n} B^n\}_{n\in\mathbb{N}}$ *inheritance functions* and the map which produces this sound space the Ξ-*operator*.

The task of creating new, more elaborate sound spaces from existing ones is thus reduced to creating families of inheritance functions and arranging existing sound spaces into a list.

3. EXAMPLES

3.1. **Time Vectors.** We next consider sound spaces of the form $(\mathbb{R}^2, \mathbb{R}^2 \xrightarrow{c} C)$. Define inheritance functions $\mathbb{R}^2 \xrightarrow{\alpha_n, \beta_n} (\mathbb{R}^2)^n$ by

$$\alpha_n(x, y) := [(x + i(y/n)), y/n) : 0 \leq i < n]$$

and

$$\beta_n(x, y) := [\underbrace{(x, y), \ldots, (x, y)}_{n \text{ times}}].$$

With these inheritance functions, we can intuitively construct many common rhythms. The family $\{\alpha_n\}$ is useful for rhythmic sequentiality, and the family $\{\beta_n\}$ for rhythmic simultaneity. For example, if $(\mathbb{R}^2, \mathbb{R}^2 \xrightarrow{b} C)$ is the sound space in which the point (x, y) represents the sound of a bongo struck at time x and resonating until time $x + y$, and if

$$r := \Xi(\beta, [\Xi(\alpha, [b, b]), \Xi(\alpha, [b, b, b])]),$$

then $r(0, 1)$ represents the "Brahms" rhythm (a simultaneous duple and triple rhythm) spanning the time interval $(0, 1)$.

3.2. **Functions of the Interval.** Rather than representing time by points of $\mathbb{R}^2$, we may use functions of the interval, $F := \{[0, 1] \to \mathbb{R}\}$. That is, we consider sound spaces of the form $(F, F \to S)$. This is useful because it permits us to elegantly realize acceler- and ritardandi. Define inheritance functions $F \xrightarrow{\alpha'_n, \beta'_n} F^n$ by

$$\alpha'_n(f) := \left[t \mapsto f\left(\frac{t + i}{n}\right) : 0 \leq i < n\right]$$

and

$$\beta'_n(f) := [\underbrace{f, \ldots, f}_{n \text{ times}}].$$

These families of inheritance functions generalize the behavior of $\{\alpha_n\}$ and $\{\beta_n\}$ because they can reproduce their effect on a time vector (x, y) by representing it by the linear function sending $0 \mapsto x$ and $1 \mapsto y$.

Functions of the interval are also useful in controlling several other parameters such as dynamics.

4. Product Inheritance Functions

Many parameters of a piece of music may be determined simultaneously by combining inheritance functions. Suppose we have two families of inheritance functions $\{A \xrightarrow{f_n} B^n\}$ and $\{C \xrightarrow{g_n} D^n\}$ and sound spaces of the form $(B \times D, B \times D \to S)$. Then we can create a new family of inheritance functions $\{A \times C \xrightarrow{h_n} (B \times D)^n\}$ by 'zipping' f and g together:

$$h_n(a, c) := [(f_n^i(a), g_n^i(c)) : 1 \leq i \leq n].$$

5. Implementation

These ideas are quite simple to implement in functional programming languages. In Haskell, for instance, we can define:

```
xi :: (Int->a->[b]) -> [b->[c]] -> (a->[c])
xi f cs a =
  concat (zipWith ($) cs bs)
  where bs = f (length cs) a
```

and the inheritance functions α' and β':

```
fAlpha, fBeta :: (Fractional a) => Int -> (a->b) -> [a->b]
fAlpha n uf = [ \t -> uf ((fi k + t) / fi n) | k <- [0..n-1] ]
  where fi = fromIntegral
fBeta = replicate
```

As a very simple example, we could then generate the Brahms rhythm with

```
l :: (Num a, Num b) => (a->b) -> [(b,b)]
l f = [(f 0, f 1 - f 0)]
brahms :: (Fractional a, Num b) => (a->b) -> [(b,b)]
brahms = xi fBeta [xi fAlpha [l,l], xi fAlpha [l,l,l]]
```

for then

```
brahms id => [(0.0,0.5),(0.5,0.5),(0.0,0.333333),
              (0.333333,0.333333),(0.666667,0.333333)]
```

where we consider each pair (t, d) as an event occurring at time t with duration d. We could also compose it with an accelerando:

```
brahms (\t->t^2) => [(0.0,0.25),(0.25,0.75),(0.0,0.111111),
                     (0.111111,0.333333),(0.444444,0.555556)]
```

These ideas are also easily implemented in Scheme:

```
(define (xi f cs)
  (lambda (a)
```

```
    (concat (map $ cs (f (length cs) a)))))
(define (f-alpha n f)
  (map (lambda (i) (lambda (t) (f (/ (+ i t) n))))
       (range 0 (- n 1))))
(define f-beta replicate)
```

provided we have defined:

```
(define ($ f arg) (f arg))
(define (range i j) (cond ((> i j) ()) (else (cons i (range (+ i 1) j)))))
(define (replicate n a) (cond ((< n 1) ()) (else (cons a (replicate (- n 1) a)))))
(define (plus-plus la lb) (cond ((null? la) lb) (else (cons (car la) (plus-plus (cdr la) lb)))))
(define (concat ls) (cond ((null? ls) ()) (else (plus-plus (car ls) (concat (cdr ls))))))
```

(Scheme, (Scheme, (Scheme...))). And then, for instance, as earlier:

```
(define (l f)
  (list (list (f 0) (- (f 1) (f 0)))))
(define brahms
  (xi f-beta (list (xi f-alpha (list l l)) (xi f-alpha (list l l l)))))
(brahms (lambda (x) x))
  => ((0 1/2) (1/2 1/2) (0 1/3) (1/3 1/3) (2/3 1/3))
(brahms (lambda (x) (* x x)))
  => ((0 1/4) (1/4 3/4) (0 1/9) (1/9 1/3) (4/9 5/9))
```

It's all also relatively simple in Mathematica:

```
xi[f_][cs___] := (Join@@MapThread[(#1[#2])&, {{cs}, f[Length[{cs}]][#]}])&
fAlpha[n_][uf_] := Function[k, uf[(k+#)/n]&] /@ Range[0,n-1]
fBeta[n_][uf_] := Table[ uf, {k,0,n-1} ]

l[uf_] := {{uf[0],uf[1]-uf[0]}}
brahms = xi[fBeta] [ xi[fAlpha][l,l], xi[fAlpha][l,l,l] ]
brahms[Identity]
  => {{0, 1/2}, {1/2, 1/2}, {0, 1/3}, {1/3, 1/3}, {2/3, 1/3}}
brahms[#^2&]
  => {{0, 1/4}, {1/4, 3/4}, {0, 1/9}, {1/9, 1/3}, {4/9, 5/9}}
```