Representing Music for Analysis and Composition

Geraint Wiggins, Mitch Harris, Alan Smaill

August 1989

Abstract

Many systems have been proposed to encode music so as to allow manipulation in a computer. We suggest that this multiplicity of representations can usefully be eliminated, and propose an *abstract* representation, which can be thought of as implicit in aspects of many previous approaches. In terms of this basic representation, we can then build up higher-level *hierarchical* representations, available for the purposes of analysis or compositional manipulation. We illustrate this approach by describing an algorithm for rhythmic analysis devloped by Steedman, and an analytic procedure developed by Ruwet, both formulated within our representation.

1 Introduction

The computer opens up a vast range of possibilities for both the composer and the analyst of music. But before the potential to manipulate musical material rapidly and intelligently can be taken up, a way must be found to allow the user to structure the raw musical material with which s/he works, which both makes musical sense to the user, and also allows computer implementation in appropriate data structures.

We believe the best approach to this general representational problem is given through the logical specification of an abstract representation. The representation should be *abstract* in that it should not commit the user to any particular representation of, say, pitch in terms of a frequency in Hertz or some particular note name. On the other hand, the representation of pitch should allow the determination of intervals from pitches, and, in general, should permit the user to determine and manipulate all the musically meaningful aspects of pitch.

It has become usual in Computer Science to make use of an abstract representation so that problems can be tackled at the right level of generality. This will make clear the common structure of problems which have often been tackled in a variety of computer languages and distinct representations, and allow partial solutions to be combined. The use of logic to specify such representations is also widely accepted.

Within Artificial Intelligence, too, research in Knowledge Representation attempts to use abstract structures to represent knowledge about objects and their relationships in the world. Logic has become the *lingua franca* of many researchers in this field. Work in this tradition on temporal logic, for example in [Allen 84], is clearly relevant to the problem of representing music.

These factors, combined with increased use of computers for analysis, composition and archiving, suggest that the time is ripe for greater exploitation of the power and expressiveness of logic. One way to contribute towards this might be the increased use of transparent logical structures in music software. This is an issue of representation: how to create a language in which it is easy to make computational objects correspond to musical objects. This issue precedes questions of implementation, for using logic as a tool it should be possible to specify an abstract representation for music which has a status independent of any particular software, computer language, or alphanumeric codes.

One advantage of such an abstract representation is that it would contribute towards the standardization, and hence compatability and clarity of music software. Music standards for communicating relatively low-level information already exist – MIDI, DARMS *etc* communications protocols. However these do not provide tools for representing higher level or novel forms of musical structure. There are so many possibilities for using computers for search, analysis of patterns or synthesis of new structures, that a unified representation which would allow different encodings to be interrelated seems called for.

In this paper, we propose the construction of such an abstract representation for music. We describe this representation, which is adequate for any note-based musical structures. Different users will wish to analyse and organise pieces of music in different ways; therefore, we describe the basis for a general framework that exploits our representation to build appropriate higher-level descriptions.

To illustrate the system at work, we have implemented analytic procedures developed by Ruwet [Ruwet 72], as used by Nattiez in various analyses of Debussy's "Syrinx" [Nattiez 75, pp 330-54]. As a second illustration we describe a cognitive model for the perception of rhythm developed by Steedman [Steedman 77]. We compare our system with other recent proposals for musical representation.

2 The Basic Representation

– Il importe de choisir un certain nombre de notions primitives *en relation directe avec le phénomène sonore – et avec lui seul –, d'énoncer, ensuite, des postulats "qui doivent apparaître comme de simples relations logiques entre ces notions . . . "* [Boulez 63, p 29]

As a first approach, let us describe a representation of the pitch and time dimensions of musical structures that we suppose given in terms of discrete notes, each of constant pitch. These notes correspond to *events* in our representation; they are the fundamental objects in the representation. The restriction to notes of constant pitch is in practice not severe; it always holds for piano music for example. While we will not discuss the details of how to represent intensity or instrumentation information here, they are treated similarly.

2.1 Representing time

The representations for time and pitch are every similar, so we simply describe the time representation here. The objects of interest are points in time, and durations (or time

intervals). The intuition behind the following definitions is that we expect to be able to describe a duration as the time interval between two points in time (hence we require a function dur); we expect to compare the durations of notes (hence we need an order $\leq$), and indeed to be able to add two durations together to get a third (hence an addition operation). So we give ourselves the mathematical machinery to do this, and some conditions so that these operations will have reasonable properties.

One concrete example would measure time in terms of number of units of some underlying pulse, say quavers, and durations in numbers of quavers. So for example the duration from beat $t1$ to beat $t2$ is $dur(t1, t2) = t2 - t1$, a duration $d1$ is less than duration $d1 + 1$, and a duration $d1$ immediately followed by duration $d2$ form together a duration $d1 + d2$. Many such concrete descriptions are possible; our object is to give a single *abstract* description.

More formally, this consists of a set of times $time$ and a set of durations $duration$, together with a mapping dur that measures the duration $dur(a, b)$ between two times a and b. To compare the durations of notes we need an order $\leq$, and to be able to add two durations together to get a third an addition operation written $+$ (not necessarily the usual addition). We take the convention that if time b precedes time a then the duration $dur(a, b)$ is negative.

Specifically, there is a function $dur : time \times time \rightarrow duration$ and there is a distinguished duration denoted by 0, a relation $\leq$ on the set $duration$, and an operation written $+$ on the duration set (with an associated inverse $-$) such that these make $duration$ a linearly ordered commutative group. [1]

We also suppose that dur is compatible with the duration structure, in that $dur(x, y) = 0 \leftrightarrow x = y$, that $dur(x, y) + dur(y, z) = dur(x, z)$, and that $dur(x, y) = -dur(y, x)$. The algebraic conditions ensure that these operations have reasonable properties. We might have been expected to provide similar operations on the set $time$, but in fact we can define the ordering on $time$ (call it $\sqsubseteq$) in terms of that on $duration$ by defining

$$t_1 \sqsubseteq t_2 \leftrightarrow 0 \leq dur(t_1, t_2).$$

We can also define in terms of what we have so far, for each time t, a function $after_t$ that gives a bijection between time points and durations by $x \mapsto dur(t, x)$; this allows us to find a time a given duration after a given time. There is a similar $before_t$ bijection.

We summarise the operations in the time representation in table 1.

2.2 Pitch, Events and Scores

The pitch representation follows the lines of that for time, so we simply mention the corresponding functions and relations. The objects of interest are pitches and (pitch) intervals, so we need sets $pitch$ and $interval$; the intervals are ordered by a relation $\leq$ and there is an addition operation. There is a measuring function $int : pitch \times pitch \rightarrow interval$, and all these are related just as in the time case.

Our basic representation of some musical structure, then, simply consists of a set of tuples, each of which we call an *event* that corresponds to a note of the music. The tuple

[1] For algebraic details, see [Mac Lane & Birkhoff 67]

sets	$time$,$duration$
functions	$dur : time \times time \rightarrow duration$
	($after_t : time \xrightarrow{\sim} duration$)
	($before_t : time \xrightarrow{\sim} duration$)
	$+ : duration \times duration \rightarrow duration$
relations	($\sqsubseteq$ (on $time$))
	$\leq$ (on $duration$)

Entries in brackets are derived notions.
The symbol $\xrightarrow{\sim}$ indicates a two-way function.

Table 1: The Time Representation

has five elements, a unique *identifier*, a *pitch* element (corresponding to the pitch of the note), a *time* element corresponding to the time of the start of the note, a *duration* element corresponding to the length of the note, and a *timbre* element that will describe timbre and intensity information. Such a description of musical notes is very natural; we emphasise again that the *pitch* , *time* and *duration* elements are taken from the appropriate abstract data types.

So the general form of event statement will have the form

```
event( Identifier , Pitch , Time , Duration , Timbre ).
```

Now, for a given musical structure, by describing the structure as a set of events over which our functions and relations may be applied we obtain a uniform way of making available most of the information needed to analyse, manipulate and create musical structures. To profit from this, we need some higher-level descriptions; this is the role of *constituents*, which we discuss in the next section.

Our approach allows both pitch and time structures to be continuous or discrete, and admits microtonal pitch structures without problem. Programs such as the Ruwet analysis method below can therefore apply without adaptation to non-classical musical systems, for example.

We have implemented various versions of these abstract data types in the logic programming language Prolog and in the functional programming language ML, where the module system is naturally appropriate to the task. Any implementation should contain constructor and destructor functions defining the datatypes, and functions to compute each of the operations defined over the datatypes.

Note that we choose to represent in the first instance (idealised) musical performances, rather than, say, musical scores — our interest is to represent music as it is experienced, rather than indirectly through some other intermediate notation. We can regard a score in this framework as a specification of a musical structure. In general, even for conventional scores, the score does not completely determine the structure, some elements being left to the interpreter; in this case we regard the score as specifying a class of structures, each of which *realises* the score. This seems to be the right way to treat scores that more radically underdetermine their interpretation, such as aleatoric scores which

may specify, for example, the notes to be played without specifying the order in which they are to be played.

3 A hierarchy of constituents

As a piece of music unfolds, its rhythmic structure is perceived not as a series of discrete independent units strung together in a mechanical, additive way like beads, but as an organic process in which smaller rhythmic motives ...function as integral parts of a larger rhythmic organisation
[Cooper & Meyer 60, p 2]

It is widely accepted that music is best described at higher levels in terms of some sort of *hierarchical structure* [Balaban 88, Buxton & *et al* 78]. These structures may play several roles, as the analyst and the composer may want to treat some given combination of note-events in quite different ways. The same chord can have quite different meanings in different styles. Our aim here is therefore to provide the framework in which such hierarchical structures can be specified, without committing the user to any particular hierarchy.

We call *constituents* the higher-level groupings of which a hierarchy may be composed. For example, we might wish to represent rhythmic groupings, or a cadence, or both at the same time, or larger groups such as a recapitulation. These are all potential constituents.

There will thus be a hierarchy where an event may appear inside a constituent, which may appear inside another constituent, and so on. We will call the events and/or constituents from which a given constituent is directly formed the *particles* of that constituent. A constituent is described by its set of particles, together with a label of its type (in the above, this could be *rhythmic_unit* or *cadence*), and a unique identifier. It may be possible and useful to assign time information to a constituent directly, rather than have this information retrieved from the particles, and we allow this possibility. This allows the efficient computation of certain temporal information, as in the TTrees approach [Diener 88].

We now illustrate how the combination of the abstract musical event structure with a hierarchy of constituents built on top can be used in practice. It will be noted that the examples make extensive use of a particular class of constituents, where the particles of the constituent naturally form a continuous melody-like succession. Other sorts of constituents are of course possible, such as those with a natural vertical structure. In a later paper we will develop the notion of constituent more fully, and explore the role that these natural subclasses play. Constituents may additionally be labelled if they fall into one of these subclasses, to indicate they are, say, melody-like. We say this information indicates the *structural type* of the constituent.

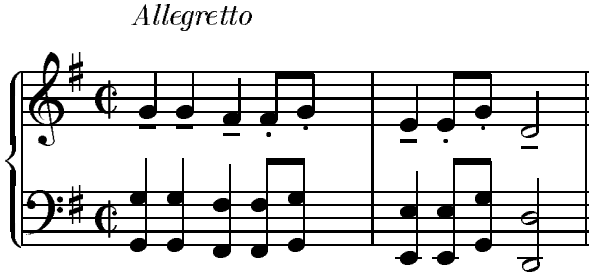

Figure 1: Mozart, Variations on "Unser dummer Pöbel meint" (bars 1-2)

4 Rhythm analysis

We show as an example how a particular applications program, for rhythmic analysis, can operate over events and constituents, creating (or suggesting) new constituents as its output. The example also illustrates the flexibility of having abstract data types: the same analysis program works irrespective of how the durations of notes in the score are transcribed. In order for these points to be made clearly, it will be helpful to explain briefly what the analysis program does.

This rhythm analysis algorithm is due to [Steedman 77], who designed it to illustrate a cognitive model of rhythm understanding. In the original program, the input would be (essentially) a list of durations[2] and the output would be a proposed time signature and phase. The program worked well over the subjects of the 48 Bach fugues, chunking the durations into bars or half-bars despite the problems presented by ties and anacrusis. This is achieved by parsing the sequence of durations according to a set of rules which pick out rhythmic primitives (*eg* dactyl, *ie* long, short, short) and accents.

We have reconstructed this program in Prolog, using our notation to output more detail than Steedman's. The program reads music presented in our basic representation and synthesises *constituents* representing the rhythmic components which are perceived according to Steedman's model. This process is illustrated below with the first two bars of Mozart's Variations on "Unser dummer Pöbel meint", representing durations with integers (see also Figure 1).

```
% event( id, pitch, start, duration, timbre ).
% pitch = [note,accidental,octave], start = # quavers from start
% duration = length in quavers, timbre as yet undefined

event( ev00, [g,natural,4], 0, 2, [ ] ).   event( ev01, [g,natural,3], 0, 2, [ ] ).
event( ev02, [g,natural,2], 0, 2, [ ] ).   event( ev03, [g,natural,4], 2, 2, [ ] ).
event( ev04, [g,natural,3], 2, 2, [ ] ).   event( ev05, [g,natural,2], 2, 2, [ ] ).
event( ev06, [f,sharp,4], 4, 2, [ ] ).     event( ev07, [f,sharp,3], 4, 2, [ ] ).
...                                        event( ev26, [d,natural,2], 12, 4, [ ] ).
```

First, the melody line is extracted automatically, in this case naïvely taking the highest note of each "chord". This generates the "melody" constituent below. Secondly, the rhythmic analysis is performed, generating constituents `st00` to `st13` and `st50` to `st54`.

[2] Represented by integers and entered 'as heard', omitting initial rests and summing tied notes.

Three of these are dactyls and the remainder contain a single note or rest. The dactyls effectively determine the largest perceived metric unit so far, and subsequent constituents reflect this chunking. The final constituent orders the four chunks, which represent the first four half-bars of the piece.

This hierarchy of constituents is adequate for this simple example, but would not be so if events were crossing the metric boundaries identified by the program (tied notes). A more sophisticated version of the program deals with this by having an extra layer of `slice` constituents called "pulses". The metric_chunk constituents then point to the pulses which, in turn, point to the underlying events. Finally, note that although the output is shown in terms of `constituent` predicates for the purpose in hand, in actual usage these would normally be further processed to produce some more readable textual output or a graphic display.

```
% constituent( id, musical_type, set_of_particles )
% time-labelling of stream (stream(t,d)) omitted for readability
constituent( st15, melody, [ev00,ev03,ev06,ev09,ev12,ev15,ev18,ev21,ev24] ).

% rhythmic units
constituent( st00, dactyl, [ev06,ev09,ev12] ).
constituent( st01, dactyl, [ev15,ev18,ev21] ).
constituent( st02, dactyl, [ev39,ev43,ev46] ).

constituent( st03, single, [ev00] ).   constituent( st04, single, [ev03] ).
constituent( st05, single, [ev24] ).   constituent( st06, single, [ev27] ).
constituent( st07, single, [ev30] ).   constituent( st08, single, [ev33] ).
constituent( st09, single, [ev36] ).   constituent( st10, single, [ev37] ).
constituent( st11, single, [ev38] ).   constituent( st12, single, [ev47] ).
constituent( st13, single, [ev51] ).

% metric chunks
constituent( st50, metric_chunk, [st03,st04] ).
constituent( st51, metric_chunk, [st00] ).
constituent( st52, metric_chunk, [st01] ).
constituent( st53, metric_chunk, [st05] ).
constituent( st54, metric_order, [st50,st51,st52,st53] ).
```

The same program will also work if the durations of the events are represented differently – by name, for example. The same section of music as above is shown below with durations represented by names and incidence times by Bar_number+Beat_number.

```
event( ev00, [g,natural,4], 0+0, crotchet, [ ] ).   event( ev01, [g,natural,3], 0+0, crotchet, [ ] ).
event( ev02, [g,natural,2], 0+0, crotchet, [ ] ).   event( ev03, [g,natural,4], 0+2, crotchet, [ ] ).
...                                                 event( ev26, [d,natural,2], 1+4, minim, [ ] ).
```

5 Similarity Analysis

As a further example, we present an implementation of general analysis procedures given in [Ruwet 72], to which the reader is referred for details of the procedure. The central idea is that various analyses of a monophonic line may be achieved by simple syntactic matching of "similar" phrases – similarity being defined in various ways. These

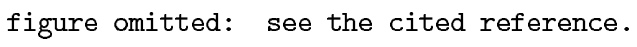

Figure 2: Analysis of "Syrinx" (Debussy) by Nattiez using Ruwet's Algorithm [Nattiez 75, p 332]

analyses correspond closely with the more conventional analyses performed on a higher, more "musical" level.

The first part of Nattiez's analysis, using Ruwet's algorithm, of Debussy's "Syrinx" for solo flute is shown in Figure 2 (see [Nattiez 75, pp 330-54]). We have transcribed the score into our representation, and are able to arrive at the same analytical conclusions as Nattiez, with respect to a subset of his similarity definition. We currently allow four kinds of similarity: Identity, Octave_Down, Near_Identity, and Loose_Octave_Down.

The first of these similarities is self-explanatory: the two associated phrases are identical. Octave_Down also requires identity, except that the pitch is transposed an octave lower in the second of the compared phrases. Near_Identity is as identity, except that the first note may be tied to a preceding note (that is to say, in our representation, the durations of the first notes of the two phrases need not be the same). Finally, Loose_Octave_Down requires that the pitches of the respective notes in the two phrases are the same, but places no conditions on the durations.

Examples of the different kinds of similarity, respectively, are A and A, A and A_1, A and A_3, and A and A_4, in the score above, taken from [Nattiez 75].

The output from our implementation of Ruwet's algorithm is of the following form. This is the statment of identity between the two phrases marked A above.

```
constituent( st000, unit, [e000,e001,e002,e003,e004,e005,e006,e007,e008,e009] ).
constituent( st001, similar( identity, st000 ),
                    [e014,e015,e016,e017,e018,e019,e020,e021,e022,e023] ).
```

6 Related Work and Conclusions

A variety of formal models of musical structures have been proposed. Our time representation allows time to be given in topological terms, so setting it apart from [Diener 88] and [Chemillier & Timis 88], who have a more restricted notion. On the other hand, our pitch representation is more flexible than the Ttrees approach, giving us a uniform approach to pitch and time.

Often these formal models describe musical structures in grammatical terms. Our work is not inconsistent with this approach – a grammar could be specified in terms of the constituent structure, and a piece of music parsed by determining automatically whether such a constituent structure could be imposed upon the underlying events. Our interest in describing the internal structure, in pitch and time *etc*, of the musical events, sets our work apart from most grammatical approaches where the notion of event is taken as a primitive with no internal structure.

Our intention is to provide the representational tools in terms of which musical analysts and composers of all sorts can perform whatever operations they want on musical structures. It is worth re-emphasising that this paper is *not* about a particular piece of software, a particular programming language or a particular type of musical analysis. What we have done is to formalise a method of representing music that makes it particularly straightforward to write programs that manipulate musical structures. We have suggested a set of abstract data structures which can be flexibly combined depending on the user's needs. We have illustrated how some simple analytical procedures can be implemented with the aid of our formalism.

A great advantage of having properly formalised abstract data structure is that much of the distracting detail (and arbitrary encoding decisions) of conventional methods could avoided. For example, it is relatively simple to map one alphanumeric encoding convention on to another *providing* both the source and the target have the same structural features. The potential exists, therefore, for true sharing of musical databases and analytical software, providing that all adhere to a uniform data structure such as the one we have suggested.

References

[Allen 84] J. Allen. A general theory of action and time. *Artificial Intelligence*, 21:121–54, 1984.

[Balaban 88] M. Balaban. A music-workstation based on multiple hierarchical views of music. In C. Lischka and J. Fritsch, editors, *14th International Computer Music Conference*, pages 56–65. Computer Music Association, 1988.

[Boulez 63] P. Boulez. *Penser la musique aujourd'hui*. Gonthier, Mayence, 1963.

[Buxton & *et al* 78] W. Buxton *et al.* The use of hierarchy and instance in a data structure for computer music. *Computer Music Journal*, 2:10–20, 1978.

[Chemillier & Timis 88] M. Chemillier and D. Timis. Towards a theory of formal musical languages. In C. Lischka and J. Fritsch, editors, *Proceedings of the 14th International Computer Music Conference*, pages 175–83, 1988.

[Cooper & Meyer 60] G. Cooper and L.B. Meyer. *The Rhythmic Structure of Music.* University of Chicago Press, Chicago, 1960.

[Diener 88] G. Diener. Ttrees: an active data structure for computer music. In C. Lischka and J. Fritsch, editors, *Proceedings of the 14th International Computer Music Conference*, pages 184–88. Computer Music Association, 1988.

[Mac Lane & Birkhoff 67] S. Mac Lane and G. Birkhoff. *Algebra.* Macmillan, New York, 1967.

[Nattiez 75] J.-J. Nattiez. *Fondements d'une sémiologie de la musique.* Union Générale d'Editions, Paris, 1975.

[Ruwet 72] N. Ruwet. *Langage, musique, poésie.* Editions du Seuil, Paris, 1972.

[Steedman 77] M.J. Steedman. The perception of musical rhythm and metre. *Perception*, 6:555–69, 1977.