



# *Music Informatics*

Alan Smaill

Feb 9th 2017

- ▶ Segmentation and grouping ctd
- ▶ A more direct representation of the musical surface
- ▶ GTTM grouping rules implemented

So far we have considered two different sorts of representation:

- ▶ acoustic signal: digitised version of audio signal
- ▶ MIDI and similar: a procedural language for operating certain forms of music-producing devices.

The first is accurate, very low level account of the sound phenomenon; the latter describes what should happen (procedure), rather than directly describing aspects of musical structure.



Alternative representations aim to describe musical structures directly in terms of their structure.

In programming languages, we distinguish between **procedural** languages (java, C, C++), and **declarative** languages (eg LISP, haskell, Prolog) – this is a related distinction, and a good analogy to musical representation.

Let's consider what a representation closer to these declarative languages for music where the musical surface is as in GTTM. There are now similar so-called **description logic** languages associated with semantic web ideas:

`http://tinyurl.com/ygtgyoz`

this is closer to database languages than what we look at now.

A starting point is to consider a note defined in terms of:

- Pitch
- Onset-time
- Duration
- Articulation
- Dynamics

There are already options here

- ▶ can articulation (staccato, slur, etc.) be assigned to an individual note?
- ▶ what units for pitch, duration etc.?

Whatever representation is used of pitch and time, we need to be able to compute some functions and relations. For example, for time, have a logical statement which corresponds to different computations:

```
get_length/3
```

```
% get_length(Time1, Time2, Length)
```

holds if `Length` is the duration between the times `Time1`, `Time2`.

Since this can be used in different ways, this also computes the time `Time2` given `Time1` and `Length`. Also need to compare times and durations for size.

A similar ability is needed for pitch.

Take notes to correspond to events, with predicate event/5:

```
event(Identifier, Pitch, Onset_time, Duration,  
      Articulation)
```

Identifier is of the form eN where N is an integer.

Pitch is a list [Name, Accidental, Octave],

*Name* is one of {c d e f g a b}

*Accidental* is one of {bb b = # x}

Onset\_time is a rational number of crotchets from start

Duration is a rational number of crotchets

Articulation is a list eg. [sl, accent]

This representation is clearly borrowed from WTM **scores**, where we give ourselves at the start start times in terms of an underlying metrical grid, and not elapsed times; it also builds in the analysis of tonal music which distinguishes between  $g\sharp$ ,  $b\flat$ .

However, for the same reason, this does give us something closer to the basics of the musical structures involved in WTM.



How to represent that some bunch of notes is considered to fit together on some musical criterion? Use another statement:  
predicate constituent/4:

```
constituent(Identifier, Property, Definition,  
            Particles)
```

where

Identifier is of the form cN where N is an integer.

Property is a logical specification of the structural properties  
of the constituent

Definition is a attached property attached by user/system

Particles is list of note-events making up a constituent.

Dynamics can then be represented using constituents where *Property = dynamic* and *Definition* is one of {pp, p, mp, mf, f, ff, crescendo, diminuendo}.

For monophonic music, connected sequence of notes is a stream:

*Property = stream*,

and *Definition* relates the stream of notes to a particular piece.

Example first bar:



## Example

```
event(e000, [c,=,1], 0/1, 1/2, [s1]).
event(e001, [a,=,1], 1/2, 1/2, [s1]).
event(e002, [f,=,1], 1/1, 1/2, [s1]).
event(e003, [e,=,1], 3/2, 1/4, [s1]).
event(e004, [g,=,1], 2/1, 1/2, [s1]).
event(e005, [b,b,0], 5/2, 1/2, [s1]).
event(e006, [g,#,0], 3/1, 1/2, [s1]).
event(e007, [a,=,0], 7/2, 1/4, [s1]).
event(e008, [g,=,0], 8/2, 1/2, [s1]).
event(e009, [d,=,1], 9/2, 1/4, [s1]).
...
constituent(c00, stream, mozart1, [e000, e001, e002, e003, ...
constituent(c01, dynamic, mf, [e000, e001, e002, e003,...
```

We will see some very simple logical rules for music described this way. These can be understood as logical statements telling us when certain statements will be true (given other statements). We can use a Logic Programming language to compute with these ruled.

For example:

```
uncle(X,Y) <- brother(X,Z) , parent(Z,Y) .
```

says that

**if** *X is brother of Z*, **and** *Z is parent of Y*,  
**then** *X is uncle of Y*

Here X,Y,Z are variables. A query with variables, if successful, returns values for any variables in the query that ensure the query is true.

**GPR 2a (Proximity)** Given notes  $n_1, n_2, n_3, n_4$ , the boundary  $n_2-n_3$  may be heard as a group boundary if

1. (Slur/Rest) the interval of time from the end of  $n_2$  to the beginning of  $n_3$  is greater than that from the end of  $n_1$  to the beginning of  $n_2$  and that from end of  $n_3$  to beginning of  $n_4$

```
grouping_rule(rule_2a, Ev1, Ev2, Ev3, Ev4) <-  
    get_transition_dur(Ev1, Ev2, Dur1), % dur from end of  
                                        % Ev1 to start Ev2  
    get_transition_dur(Ev2, Ev3, Dur2),  
    get_transition_dur(Ev3, Ev4, Dur3),  
    gtdur(Dur2, Dur1),  
    gtdur(Dur2, Dur3).
```

**GPR 2b (Proximity)** Given notes  $n_1, n_2, n_3, n_4$ , the boundary  $n_2 - n_3$  may be heard as a group boundary if

1. (Attack-Point) the interval of time between attack points of  $n_2$  and  $n_3$  is greater than that between the attack points of  $n_1$  and  $n_2$  and that between the attack points of  $n_3$  and  $n_4$ .

```
grouping_rule(rule_2b, Ev1, Ev2, Ev3, Ev4) <-  
  get_attackpt_dur(Ev1, Ev2, Dur1),  
  get_attackpt_dur(Ev2, Ev3, Dur2),  
  get_attackpt_dur(Ev3, Ev4, Dur3),  
  gtdur(Dur2, Dur1),  
  gtdur(Dur2, Dur3).
```

**GPR 3a (Change)** Given sequence of notes  $n_1, n_2, n_3, n_4$ ,  $n_2-n_3$  can be heard as a boundary if

1. (Register)  $n_2-n_3$  is a larger pitch interval than both  $n_1-n_2$  and  $n_3-n_4$

```
grouping_rule(rule_3a, Ev1, Ev2, Ev3, Ev4):-  
    get_abs_pitch_int(Ev1, Ev2, Int1),  
    get_abs_pitch_int(Ev2, Ev3, Int2),  
    get_abs_pitch_int(Ev3, Ev4, Int3),  
    gtint(Int2, Int1),  
    gtint(Int2, Int3).
```

These two rules apply to our small example:



Rule 2a allows a boundary after the 4th note;  
rule 3a allows a boundary after the 5th note.



Here implement a simplified version of the rule.

**GPR 3d (Change)** Given sequence of notes  $n_1, n_2, n_3, n_4$ ,  $n_2-n_3$  can be heard as a boundary if

1. (Duration)  $n_2-n_3$  involves a larger change in duration than both  $n_1-n_2$  and  $n_3-n_4$ ,

```
grouping_rule(rule_3d, Ev1, Ev2, Ev3, Ev4) <-  
  get_length(Ev1, L1),  
  get_length(Ev2, L2),  
  get_length(Ev3, L3),  
  get_length(Ev4, L4),  
  not eqdur(L2, L3),  
  eqdur(L1, L2), eqdur(L3, L4).
```

**GPR 6 (Parallelism)** *Where two or more segments of the music can be construed as parallel, they preferably form parallel parts of groups.*

This is more complex. Need to decide

- ▶ what counts as parallel
- ▶ how to search for parallel instances

eg, only look for patterns of 4 consecutive notes for match;  
if match is found, try to extend to longer match.

Start with first four notes; then try notes 2–5, etc.

Some notions of parallelism:

- ▶ Identity; corresponding notes same pitch and duration
- ▶ Long identity; identity, except first note differs in duration
- ▶ Transpose; shifted by constant pitch interval, durations same
- ▶ Loose Transpose; shifted by constant pitch interval, ignore durations
- ▶ Tonal transpose (using note names without accidentals)

Each of these is easy to program;  
There is a **lot** of choice as to what is put here.

At this point it is clear that rules 2 and 3 to start with generate more possible boundaries than make musical sense.

One possibility is to allow user to choose between groupings on the basis of evidence (eg how many rules apply), and use this to generate a grouping analysis.

The following images show initial applications of rules 2, 3; and a possible analysis using some of these with other rules (parallelism is rule 6).

The final analysis gives us a possible parsing of the musical surface into a parse tree, justified by the rules. The choice among the preference rules is made here interactively by the analyst.

## Given musical surface



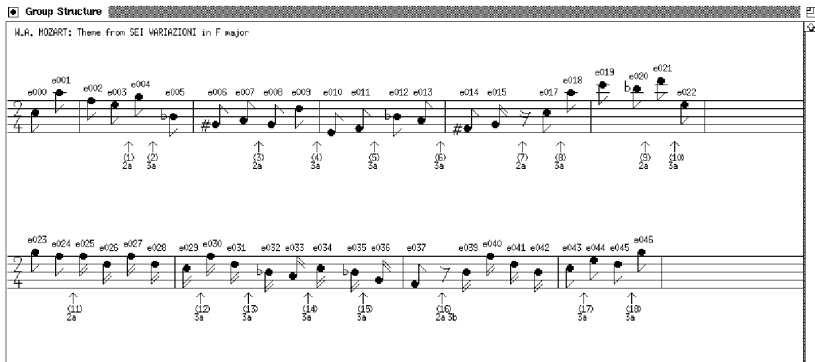
*mf*

6 *p*

11 *cresc.* *f*

**Group Structure**

W.A. MOZART: Theme from SEI VARIAZIONI in F major

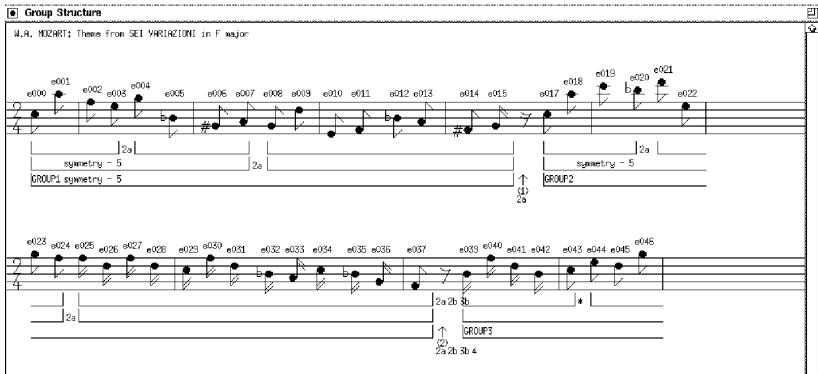


Measures: e000, e001, e002, e003, e004, e005, e006, e007, e008, e009, e010, e011, e012, e013, e014, e015, e017, e018, e019, e020, e021, e022, e023, e024, e025, e026, e027, e028, e029, e030, e031, e032, e033, e034, e035, e036, e037, e039, e040, e041, e042, e043, e044, e045

Fingerings: (1) 2a, (2) 3a, (3) 2a, (4) 3a, (5) 3a, (6) 3a, (7) 2a, (8) 3a, (9) 2a, (10) 3a, (11) 2a, (12) 3a, (13) 3a, (14) 3a, (15) 3a, (16) 2a 3a, (17) 3a, (18) 3a

**Group Structure**

W.A. MOZART: Theme from SEI VARIAZIONI in F major



e000 e001 e002 e003 e004 e005 e006 e007 e008 e009 e010 e011 e012 e013 e014 e015 e017 e018 e019 e020 e021 e022  
 symmetry - 5 symmetry - 5 symmetry - 5  
 GROUP1 symmetry - 5 GROUP2  
 (1) 2a

e023 e024 e025 e026 e027 e028 e029 e030 e031 e032 e033 e034 e035 e036 e037 e039 e040 e041 e042 e043 e044 e045 e045  
 2a 2b 3b \*  
 GROUP3  
 (2) 2a 2b 3b 4

A grammar is said to be **context-free** if it has the property that each grammar rule looks like the rules from our simple musical grammar:

line --> bar1, bar, bar, bar4.

in that the left-hand side of each rule consists of a **single** non-terminal symbol (the right-hand side is any combination of terminals and non-terminals).

Such grammars allow efficient parsing algorithms.



**Context-sensitive** grammars allow more complicated rules, where the left-hand side prescribes a context for the use of the rule:

`[a], line, [b] --> [a], weird, [b]`

Parsing is a harder task for such languages.

The grouping rules are **not** context-free (because of, eg parallel rule). Intuitively, a context-free grammar allows us to recognise when we have an instance of a grammatical class (here, a chunk) by looking at some **consecutive** instances of other classes.

Given a context-free grammar, it is not necessary to look outside the chunk we are analysing, ie to look at the context. If we have such a grammar, the parsing task is relatively easy. The parallelism rule introduces long-range dependencies – in principle, the parallel occurrences can be arbitrarily far apart.

This is incompatible with a context-free grammar for grouping.

- ▶ declarative note-based representation
- ▶ implementing grouping rules
- ▶ how use in practice?