50th sempre: Society for Education, Music and Psychology Research Anniversary

MUSIC & SCIENCE

Research Article

Music & Science
Volume 5: 1–20
© The Author(s) 2022
DOI: 10.1177/20592043221085659
journals.sagepub.com/home/mns
SAGE

funCode—Versatile Syntax and Semantics for Functional Harmonic Analysis Labels

Markus Lepper[1], Baltasar Trancòn y Widemann[1,2] and Michael Oehler[3]

Abstract

Traditional harmonic analysis annotations can be represented in a computer model of a piece of music by plain text strings. But whenever automated processing like analysis, comparison or retrieval is intended, a formal definition is helpful. This should cover not only the syntactic structure, but also the semantics, i.e. the intended meaning, and thus adheres to the technique of *mathematical remodelling* of existing cultural phenomena. The resulting models can serve as a basis for automated processing, but also help to clarify the communication and discussion among humans substantially. This article proposes such a definition in four layers, which address different problems of encoding and communication: (a) relation of symbol sequences to staff positions, (b) combining functions, (c) chord roots, and (d) interval structure and voice leading. Only one of them is specific to functional (Riemannian) theory and can possibly be replaced to represent scale degree theory. The proposal is configurable to different interval specification methods and open to localisation. Syntax and semantics are defined by precise mathematical means, borrowed from computer science, and thus are unambiguously documented.

Keywords

Formal semantics, context free language, harmonic analysis, Riemannian analysis, computational thinking

1 Introduction

Symbol sequences representing harmonic analysis results can be attached to computer models of music as a mere auxiliary device for retrieval, analysis and comparison. Furthermore, in (historic or contemporary) documents from published music theory, such sequences may have a value on their own, as creative utterance by their author. In either case, automated processing requires clarification and specification of syntax and semantics of these symbol sequences and their different variants, by applying mathematical modelling. Such a clarification can also be helpful to add precision to communication and discussion among humans. The funCode proposal defines a framework in four layers, which address different problems of encoding and communication: relation of symbol sequences to staff positions, simultaneously sounding functions, chord roots, interval structure, and voice leading. funCode models the *functional* style of harmonic theory of Western music. Only the third layer must be exchanged for an adaptation to scale degree theory; selected layers can possibly be adapted to other cultures, whenever these employ the modelled concepts.

1.1 Harmonic Analysis Methods and Symbol Systems

Modern harmonic analysis of Western music began in 1726 with Jean-Phillipe Rameau ([1722]1965) and with the idea that particular patterns of interval structures and of chord sequences may have a characteristic effect on the listener, independent of most other aspects of a musical work like voice leading and instrumentation, and especially independent from transposition. Theories of musical harmony in the widest sense are already known from the times of antique Greek philosophy. But only when the keyboard became the leading instrument in practice and theory, and printing and publishing opened access to a wide range of diverse musical literature,

[1] semantics GmbH, Berlin, Germany
[2] Nordakademie Elmshorn Germany
[3] Universität Osnabrück Germany

Corresponding authors:
Markus Lepper, semantics GmbH, Berlin, Germany.
Email: post@markuslepper.eu

Michael Oehler, Universität Osnabrück, Germany.
Email: moehler@uni-osnabrueck.de

could a debate about the harmonic structure of complex and elaborate compositions be conducted on an objective basis.

Various theories of musical harmony were proposed in the following centuries, with very different intentions: a theory can act as a mere description of a particular practice, or it can try to find reasons (based in physics or physiology) why a particular practice is sensible, or it can try to name and explain effects caused in the recipient's mind; or, in the other extreme, a harmonic theory can act as a prescription, as a 'cook book' for correct accompaniment or effective composition, in a particular style. Most of the historic theories serve a certain mixture of these use cases.

A first milestone in the chain of reactions to Rameau was set by Weber (1817), who examined the triads on every step of the diatonic scale, in major and minor mode, and invented the *roman numeral notation*. Hugo Riemann started in 1877 with a large catalogue of possible chord interval structures and an even larger one of possible distances and relations between pairs of chords. (Riemann, 1877; Engebretsen, 2011) After many years filled with the tedious work of refactoring his system again and again, he finally arrived at the *functional theory* of harmony, which derives every chord from the three basic functions tonic, dominant and subdominant (Riemann, 1918). There were many others going also in this direction, even earlier, but it was him who found, at last, the clearest formulation and most comprehensible symbol system.

In the first half of the twentieth-century in particular, numerous further theories evolved, which can be strictly divided into the 'roman numeral' tradition on one side (also called 'scale degree theories', German 'Stufentheorie') and the 'functional' approach on the other. In more recent times, new theories have arisen beyond this dichotomy, based e.g. on the notion of pitch classes modulo enharmonics (Forte, 1973; Rahn, 1980), or based on even more advanced notions from mathematics like topologies (Vogel, 1975; Mazzola, 1990; Tymoczko, 2011; Cambouropoulos et al., 2014).[1] But in the traditional approaches, that dichotomy is still relevant, especially in education.[2]

In all these theories the central means for discussing, explaining and analysing is some *specially designed symbol system*. This is given by a collection of rules governing how to create short-cut symbols for chords, their interval structures and their progressions, how to combine these symbols, and what these combinations should express. A first use case is to attach these symbols to selected time points in a sequence of notated music, which means that at these time points the harmonic situation corresponds to or is explained by the symbol. This is called a *harmonic analysis* and a symbol attached to a particular time point is a *(harmonic or functional) label*. A quite different use case is to employ a stand-alone sequence of such labels to mean an abstract sequence of harmonic situations or to represent *any* concrete fragment of music that matches it.

Each such theory comes with its own symbol system. Nevertheless there are pedigrees, variants, common subsets and families of systems, which treat many details in a similar or even identical way.

1.2 Possible Semantics of Harmonic Analysis Label Sequences

In practice there are very different kinds of semantics that can be assigned to a harmonic analysis label sequence, no matter whether functional symbols or roman numerals are used:

S-1 Making a psychological statement about the reception of a particular piece of music by 'a listener', like:
'This sequence of chords will be recognised as $(\mathbb{D}^7)\ s^N\ D^7\ t$, but not before hearing the third chord.'[3]

S-2 Describing an (assumed) syntactical pattern, to make different scores (possibly in different keys) comparable, as in: 'The sequence $(\mathbb{D}^7)\ s^N\ D^7\ t$ appears at these positions: …'

S-3 For talking about concepts in general, like in 'Beethoven in his early works prefers the sequence $\mathbb{D}^{\substack{7\\5>}}\ D^7\ t$ as nearer to the current tonic key than $\mathbb{D}^7\ D^7\ t$.'

S-4 For generating a concrete set of pitch classes. For example, when applied to the tonal key C, the first above-mentioned coding will produce this sequence of sets of pitch classes

$$\langle\{a\flat, c, e\flat, g\flat\}, \{d\flat, f, a\flat\}, \{g, b, d, f\}, \{c, e\flat, g\}\rangle$$

and this sequence of bass tones:

$$\langle a\flat, f, g, c\rangle$$

S-5 In the tradition of the older 'figured bass' notation, the interval numbers in the superscript of the functional symbol can be meant to indicate *voice leading*: two different numbers in the same stack position in subsequent superscripts indicate a change from the first to the second interval, executed by some voice of the score.

There are coarse as well as subtle differences between these use cases: 'the listener' in case S-1 can be (S-1-0) a theoretical construct, an 'ideal listener'. Or it can simply stand (S-1-1) for the personal feeling of the author, without further claims. Or they can be (S-1-2) theoretic representatives of particular social groups with dedicated training, like 'the average Telemann addict' or 'the hip-hop recipient'. Or they may be (S-1-3) concrete people from an empirical study. The real meaning of the symbol sequence is slightly different in each of these cases.

Semantics S-2 can be useful with different degrees of precision, for instance, one may want to address chord sequences without specifying the inversions.

Semantics S-3 can be relevant for encoding and automated processing e.g. when comparing, documenting and analysing analyses as such, i.e. works in music theory. It could be a fruitful attempt to translate the (approximately) 50 different publications on the harmonic structure of the first twelve measures of *Tristan and Isolde* into one format, strong enough to capture all intended semantics, and apply automated analysis.

Semantics S-4 has most of its relevance on the meta-level of discourse and practical work-flow: it shows which pitch classes are not covered, i.e. which are in the notes but not considered relevant by the author of the analysis, or, conversely, which are not present in the score but in the effective harmony, according to that author. Thus it can be used for correctness checks by author, editor, reviewer and reader.

Only the semantics of kind S-4 are currently computable by machines. It is implemented by funCode for the functional style of musical analysis, as it is done by Nápoles López & Fujinaga (2020) for the roman numeral style.

The differences between the semantics S-1 and S-4 are the widest: in S-1 the terms `DD` and `SP` describe two different functional derivations, implying different inner mental representations induced by listening. In S-4 they simply deliver identical sets of pitch classes. Similar with **`D/79+`** vs. **`Sp6+`**, with **`s6-`** vs. **`sG3_`**, etc. Each of these cases is a typical example of identical 'reference' ('denotation'/'significance'/ 'Bedeutung'), but different 'sense' ('meaning'/'Sinn'), as discussed by Hyer (2011, pg. 121*ff*) referring to Frege. This corresponds to the difference between 'measuring […] as an empirical fact' and 'as an intuited chain of intuitions', strongly emphasised by Lewin ([1987]2007, pg. 18).[4]

1.3 Intended Use Cases

In the tradition of mathematical *re*-modelling (Lepper, 2021), our proposal does not invent anything new, but only tries to unify and simplify a family of historically evolved annotation styles, which are currently in wide use, and give them precisely defined mathematical semantics—to make them accessible for automated processing and to clarify their semantics unambiguously for human discourse. funCode is thus a contribution to *practical data processing*, not to music theory, but based on the theories of computer languages and harmonic analysis.

Even nowadays authors must still explicitly clarify their labelling system. The first footnote in the article by Heetderks (2015) (which is taken in the following as an example of a recent, typical and arbitrarily chosen publication using scale degree theory) starts: 'In this article, major triads will be indicated by …' In the context of more and more automated processing by digital systems, but also for human discussion, precisely defined but versatile and modifiable encoding standards are desirable. Therefore the authors of the recent 'Annotated Beethoven Corpus' (Neuwirth et al., 2018) added a formal definition of the syntax of their scale degree labelling system. For applications of funCode in the human discourse see the discussion of the abbreviation "`Gr`" in section 2.10 and the naming problems in Table 5.

The formalisation of funCode is published in a technical report (Lepper et al., 2022) as a Prolog program which at the same time is executable and serves as a formal specification of syntax and semantics. The adaptability of this basic framework is intended to map different labelling systems, thus making them accessible e.g. to automated translation and comparison, and clarifying their semantics in human discussion.

The implementation is also maintained as open source, currently on `sourceforge.net`. It allows funcode source texts to be parsed and evaluated into (sets of) coordinates in the Euler space. These results could e.g. be further reduced to 12-tone pitch classes and fed into existing retrieval engines to search for harmonic constructs which are more conveniently written in the functional style, like the example **`(D)sG3_ D7 t`** from above.

As a by-product, thousands of existing analyses in the European functional tradition can possibly be connected to contemporary analytical tools (from Neo-Riemannian and transformation theory) semi-automatically by aligning them manually to the precisely defined funCode language and processing them as described in detail in section 2.8.

The implementation additionally provides a layout algorithm to generate conventional two-dimensional rendering and an experimental LaTeX back-end, see section 2.2.

Integrating harmonic labels into a computer model of music is a non-trivial task not yet addressed, and is briefly discussed at the end of section 2.1.

A (semi-)automated annotation of music scores with functional symbols is currently out of reach. Nevertheless funCode is a first contribution: before attempting an automated translation, a precise and unambigious specification of a target language is required.

1.4 Design Principles of the funCode Approach

Not intending to invent something new, nevertheless in mathematical re-modelling always a certain 'clean-up' takes places. Any language found in cultural practice has evolved through history, but has not been designed ex ovo as a 'Domain Specific Language' with properties defined by the contemporary professional mathematical theory of language and parsing. Consequently, there are always some 'rough edges' which should not be taken over but straightened out, in particular, because these are often only small peculiarities that cause considerable damage to compositionality, general applicability, easier parsing, etc. Most of the corrections to apply are *canonical continuations*: they remove a particular restriction which evolved in practice as 'professional blinker' but which is not necessary from the mathematical viewpoint.[5] 'Computational thinking' (Broy, 2011) means to learn from automated processing for the preciseness and handiness of human communication.

Thus some basic properties have been clearly marked as indispensable:

D-1 Transposition-invariant symbols: every identical harmonic pattern must deliver identical symbol sequences, e.g. appearing in a key of c major, c sharp major and c flat major. The harmonic substance which shall be encoded must be readable independently from the transposition and always completely explicit.[6]

D-2 Syntax and semantics must be easily readable and writable by humans and machines.

D-3 Semantics must be recognised with minimal context information. (Nevertheless, little dependency may be allowed for ergonomic reasons. For example, **`D7`** always means a *minor* seventh, an abbreviation following convention and practical usability, but **`T7-`** and **`T7+`** must always be qualified explicitly.)

D-4 Localisation and further adaptability: every harmonic notation system has its merits, and many scholars have been used to a particular system for decades, which they want to continue to use. Since in funCode all possible adaptations are formalised, complete automatic translation between these variants is feasible.

1.5 The Layered Syntax of funCode

All funCode specifications are organised by four hierarchically layered dimensions of addressing. They model the concrete practice when adding analytical labels to a piece of music, which has remained largely unchanged since its historic beginnings: the top part of Figure 1 shows the (simplified) music in staff notation. Below are the added labels (regardless of whether roman numerals or functional theory). The very last line (in teletype font) shows the funCode encoding of that complete analysis. The arrows and numbers in gray are not part of the analysis notation but added to indicate the dimensions of its organisation:

L-1 On the top level, one can have a vertical stack of *tracks*. In practice, the existence of more than one track can have very different meanings, see section 2.1 below. This is called *conventional two-dimensional arrangement (C2DA)* in the following and has existed from the very first beginnings till today, see Figure 3, cited from Heetderks (2015), a recent publication.

L-2 Internally, each track is a horizontal sequence of labels. The graphical position establishes the relation between each label and a time point in the music.

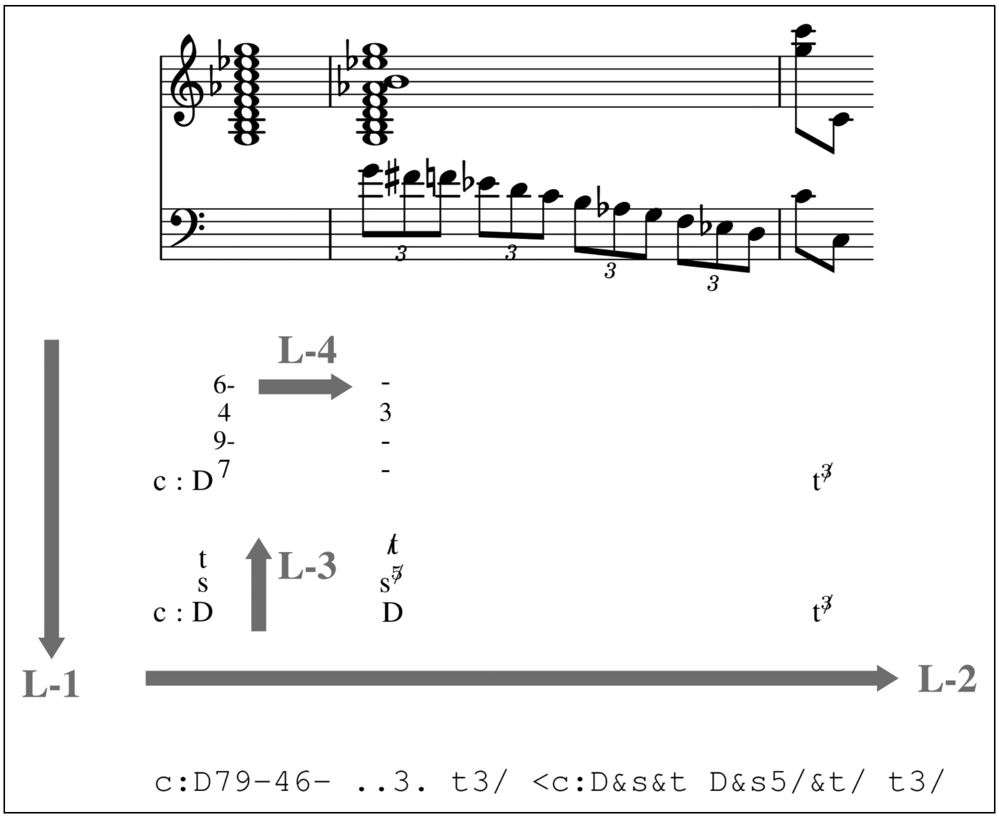


Figure 1. Typical appearance of a harmonic analysis by labelling. The top part shows a simplified staff notation of the music under consideration (Mahler, II/1, m.327ff). Below are two tracks in conventional two-dimensional arrangement (C2DA), aligning the functional labels vertically with the score positions. The gray layer is not part of the notation, but a comment on it, showing the four axes of semantics and their representation by graphical arrangement: (L1) tracks, (L2) score positions, (L3) simultaneous chords, and (L4) voice leading. The bottom line shows both tracks as funCode source text.

L-3 Each horizontal position in the track can carry zero to many function symbols. A label combining more than one function symbol stands for a *compound chord* (a case more frequently found with functional symbols than with roman numeral analysis).[7]

L-4 The finest coordinate models the voice leading internal to the same chord: few voices change their pitch from on score position to the next, without altering the chord's basic function.

Table 1 defines the syntax of all four layers of funCode *completely*; only the structure of the track names and of the interval modifiers are pluggable and appear as parameters T and M. The syntax is given as an 'extended' context free grammar (Wikipedia contributors, 2021), where on the right side regular expressions (Wikipedia contributors, 2022) may appear. Their operators $\_ \mid \_$, $\_^?$, $\_^*$ and $\_^+$ have the usual meaning of alternative, option, and arbitrary and non-empty repetition, respectively.

The syntax and semantics of all four layers of the funCode architecture will be explained step by step in section 2.

1.6 Fundamental Entities of funCode Semantics

Figure 2 shows the fundamental semantic entities which result from parsing and evaluating a funCode label sequence as UML classes (UML, 2022). Please note that these correspond not always to concrete class definitions of implementation, but exist on the conceptual level; in the current implementation (Lepper et al., 2022) they are realised by term compounding or by facts in the Prolog data base. (This section is for orientation only, all these entities are explained in detail in the next section.)

Each combination of **Track** and **ScorePos** identifies at most one **Sum** node, realising the axes L-1 and L-2 as described in the preceding section and in Figure 1. Each **Sum** node represents the axis L-3 and contains one or more simultaneous **Chord** nodes. Each of these requires a **RootExpr** and may carry **ExplicitInterval**s. These in turn must have a **Number** (= their traditional name, encoded numerically) and may have a **Modifier** (to indicate major, minor, augmented, or diminished size etc.). The sequence of intervals is *ordered* (in the funCode source text horizontally, in C2DA vertically), which allows intervals to be *inherited* from a predecessor chord with an identical root, along with axis L-4. The pitch classes which do sound for a particular label result from the explicit intervals in the source text and the rules for default intervals. Each **Sum** node **liesIn** a ***Context***, which is either a **Track** or a **RelativeSection**, and which is used to resolve the pitch of the relative root expression. Each ***Context*** also has a context, except the top-most track—the graph of the 'liesIn' relation forms a tree, which is not easily expressible in UML.

Each **RelativeSection** is **relativeTo** either a **Sum** (with only one **Chord**!) or a **Virtual**. The latter consists only of a **RootExpr** and means a function *expected* in a context, but not realised by any sounding note in the music.

The (optional) tonal center of a track, the root symbols, the explicit intervals, and all properties derived by their summation, evaluate into the **Euler** data type.

2 The Four Layers of funCode

2.1 Top-Level: Track Building and Horizontal Score Positions

As mentioned above, doing harmonic analysis can be seen as adding labels under the music, at selected time points.

Table 1. Complete syntax of all funCode layers.

$$
\begin{array}{l}
[T, M] \\
\mathit{analysis} ::= \mathit{trackTitle}^?(\mathit{tonicCenter}\ \mathbf{:})^?(\mathit{funSeq} \mid \mathbf{<}^*\mathbf{\{}\ \mathit{analysis}\ \mathbf{\}})^* \\
\qquad\qquad (\mathbf{<}^+\mathit{analysis})^? \\
\mathit{trackTitle} ::= \mathbf{"}\ T\ \mathbf{"} \\
\mathit{tonicCenter} ::= \mathit{whiteKey}\ (\mathbf{b} \mid \mathbf{\#})^*\ (\mathbf{,} \mid \mathbf{'})^* \\
\mathit{whiteKey} ::= \mathbf{A} \mid \mathbf{B} \mid \mathbf{C} \mid \mathbf{D} \mid \mathbf{E} \mid \mathbf{F} \mid \mathbf{G} \\
\mathit{funSeq} ::= (\ \mathit{funPred}^?\ \mathit{funSound}\ \mathit{funSucc}^? \\
\qquad\qquad \mid \mathit{funPred}\ \mathbf{[}\mathit{rootAndMode}_{\mathrm{N}}\mathbf{]}\ \mathit{funSucc}^? \mid \mathbf{[}\mathit{rootAndMode}_{\mathrm{N}}\mathbf{]}\ \mathit{funSucc} \\
\qquad\qquad \mid \mathit{funSounds} \mid \mathbf{-} \mid \mathbf{\sim} \mid \mathbf{>} \mid \mathbf{!})^+ \\
\mathit{funPred} ::= \mathbf{(}\mathit{funSeq}\ \mathbf{:}^?\mathbf{)} \\
\mathit{funSucc} ::= \mathbf{(:}\mathit{funSeq}\mathbf{)} \\
\mathit{funSound} ::= \mathit{rootAndMode}\ \mathit{suppress}^?\ \mathit{intervals}^? \mid \mathit{intervals} \\
\mathit{funSounds} ::= \mathit{funSound}\ (\mathbf{\&}\ \mathit{funSound})^* \\
\mathit{rootAndMode} ::= (\mathbf{S} \mid \mathbf{s} \mid \mathbf{T} \mid \mathbf{t} \mid \mathbf{D} \mid \mathbf{d})\ (\mathbf{P} \mid \mathbf{p} \mid \mathbf{G} \mid \mathbf{g} \mid \mathbf{D} \mid \mathbf{d} \mid \mathbf{S} \mid \mathbf{s})^* \\
\mathit{suppress} ::= \mathbf{/} \mid \mathbf{//} \\
\mathit{intervals} ::= (\ \mathbf{.} \mid \mathit{intervalDecorated})^+ \\
\mathit{intervalDecorated} ::= \mathit{interval}\ (\mathbf{/} \mid \mathbf{,} \mid \mathit{iModifier}^?\ \mathbf{\_}^?\mathbf{\hat{}}^?\mathbf{\_}^?)^? \\
\mathit{interval} ::= \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \mathbf{4} \mid \mathbf{5} \mid \mathbf{6} \mid \mathbf{7} \mid \mathbf{8} \mid \mathbf{9} \mid \mathbf{10} \mid \mathbf{11} \mid \mathbf{12} \mid \mathbf{13} \mid \mathbf{14} \\
\mathit{iModifier} ::= M
\end{array}
$$

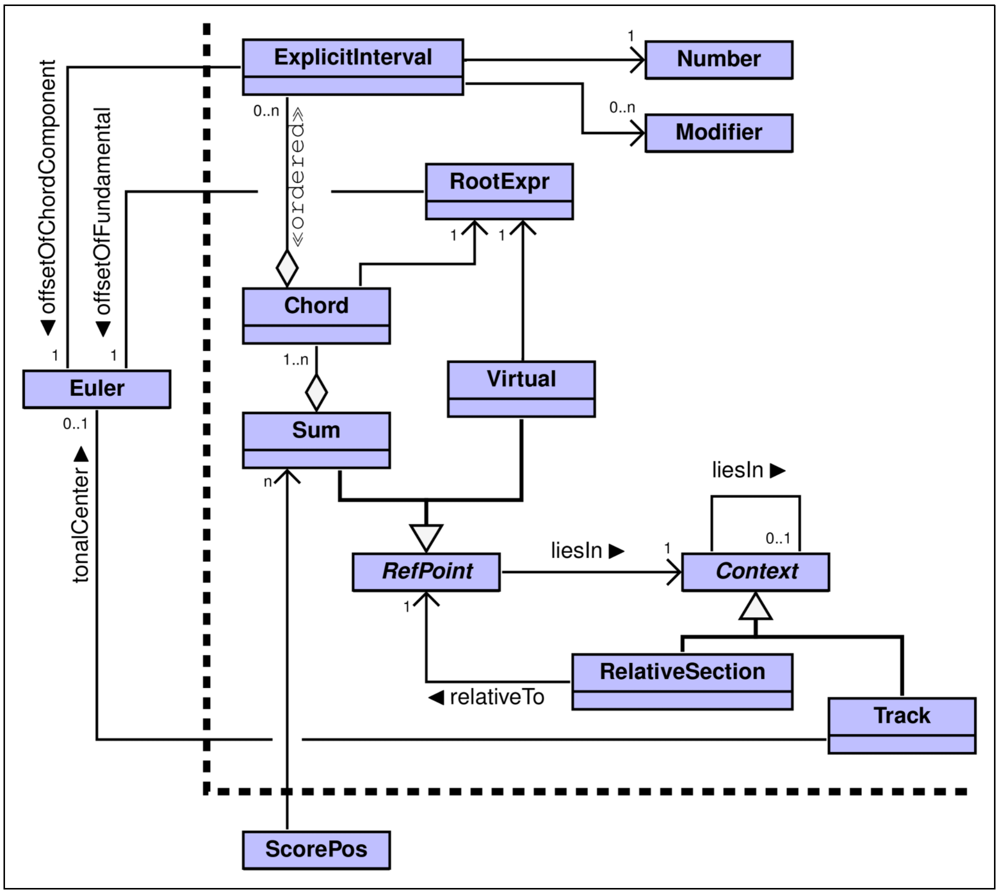
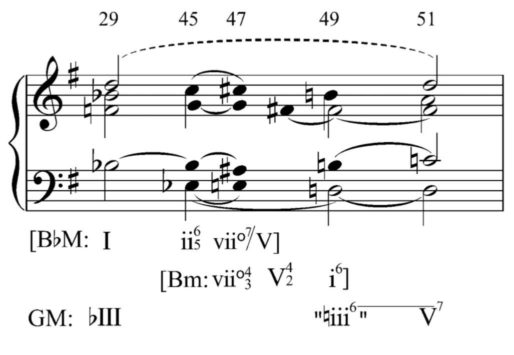


Figure 2. Most important semantic entities seen as UML Classes. Simplified diagram: attribute definitions are not shown; boxes without attribute part are mere scalar data types or simple interfaces.

(a)


29 45 47 49 51
[B♭M: I ii$^6_5$ vii°^{7}/V]
[Bm: vii°$^4_3$ V$^4_2$ i^6]
GM: ♭III "♮iii^6" V^7

(b) B:T S56+ DD/79-

H:D/7_9- .8 t3_

G:tP Tg3_ D7

(c) `B♭M:I ii56 >vii°7/V <{Bm:vii°34 V24 i5}<<GM:♭III ˜ ˜ ♮iii6 V7`

(d) `B:T S56+ >DD/79- <{H:D/79- .8 t3_}<< G:tP ˜ ˜ Tg3_ D7`

Figure 3. Analysis from Schubert D 960 by Heetderks (2015), using sub-tracks. (a) = staff notation and roman numeral analysis from the publication; (b) = translation into functional style; (c) and (d) = application of the funCode L-1 linear encoding to (a) and (b).

Assume all these time points are numbered consecutively, starting with 1. By which means this numbering is established is out of scope of funCode—in C2DA it is done by mere vertical juxtaposition. The relation from labels to the music is established only by this index, called *score position* in the following: the source text of a funCode expression is parsed from left to right and the recognised labels are consecutively assigned to the score positions. The next position to be assigned at a particular parsing state, i.e. when interpreting a particular position in the funCode source text, is called *current score position* in the following.[8]

An analysis is often organised in several vertically stacked and horizontally extending *tracks*. Tracks can be employed for very different purposes:

T-0 To model psychological ambiguity during 'modulations' in one single (ideal) listener. This is the most frequent use case in the conventional textbook analysis.
T-1 For describing different parts of the music which sound simultaneously, e.g. one track describing an organ point and the track above the changing upper voices.
T-2 To document different interpretations by different listeners, found empirically or proposed in theory.[9]
T-3 To represent different opinions by different scholars, for comparison and discussion, or
T-4 in school lessons: different solutions by different pupils.

Printing and hand-writing normally use different vertical positions for different tracks (similar for interval stacking and voice leading, as described later). But this is tedious and possibly ambiguous, e.g. when reusing physical lines on the paper or at the line and page breaks. Conversely, the linear funCode syntax (see Table 1) is always unambiguous. It uses the characters "`>`", "`<`", "`{`" and "`}`":

The reserved character "`>`" sets a tabulator stop at the current score position. The construction of each track starts with one tab stop implicitly set at its starting score position.

The braces "`{`" and "`}`" open a new track which starts at the current score position. Prepending one or more "`<`" characters go back one or more tab stops to start this new track. After the closing brace, the interrupted track will be continued as if the braced source text were not present. From the source text perspective, the second track can be called a *sub-track* of the first, its *parent track*. The sub-track can inherit some *context information*, in particular, the current tonic reference point. The grammar in Table 1 shows that a tonal key (denoted by the non-terminal *tonicCenter*) can be given at the start of any (sub-)track explicitly, thus overriding the inherited value.

One or more "`<`" *not* followed by a brace also start a sub-track, but with no means of continuing the interrupted one, which is thus considered to be complete. This is a standard use case with modulations, i.e. shifts of the current tonal centre.

Figure 3 shows the transcription of a typical analysis from a recent publication: in the upper part is the staff and the roman numeral analysis as published by Heetderks (2015). Below is the same analysis rewritten by us in a functional style (using German pitch names), followed by both versions as a linear source text using the L-1 track syntax of funcode. (Only the functional part is really formalised by funCode; applying L-1 to roman numeral analysis is done here informally and left to future work.)

Each track can be given a title, which currently is any sequence of characters not containing the double quote '`"`'. This title could encode the roles of and a hierarchical order among the tracks, but this is currently not defined by funCode.[10]

When an analytic annotation is already encoded as a computer model, e.g. as MusicXML or MEI, then tracks and labels are already separated into different 'XML elements' of the encoding. In this case the top-level L-1 funCode operators are not needed, but parsing and evaluation of the other layers L-2 to L-4 is applied directly to these elements' contents. Then all data flow by inheritance (from parent track to sub-tracks, but also between adjacent chords, see below) must be provided for accordingly. These are non-trivial problems not yet addressed by funCode. Similar problems arise with the more abstract proposal by Hentschel et al. (2021), in which each piece of information is anchored at some particular chord object and contextual cross-references as required by the funCode layers L-1 and L-4 are not foreseen.

2.2 Details and Pitfalls With Tracks and Score Positions

While syntax and semantics of the L-1 layer seem rather simple, indeed subtle pitfalls and problems had to be solved by taking appropriate design decisions. Figure 4 shows examples (using dummy contents).

We want to give precise semantics to the common pen-and-paper practice. In C2DA a parent track is found by starting at the beginning point of the sub-track and searching upwards for the first track which starts earlier (= C2DA-subtracking). Thus a sub-track may not start earlier than its parent track which (possibly) defines a new tonic centre, to be recognisable in C2DA; therefore in funcode no tab stops earlier than its beginning are inherited by a sub-track. In the first example from Figure 4 this prevents the source variant "`{e:E F G << J K }`" which would yield the same score positions, but for "`J K`" the tonal context "`e:`" in the source text versus "`a:`" by C2DA-subtracking.

Tab stops later than the beginning *could* be inherited if they are ignored as long as they are not yet overtaken by the current score position. But this would make the meaning of the "`<`" operator context-dependent, which can be confusing. In the tracks defined by "`A B >C D <<{E < X}`" versus "`A B >C D <<{E F G < X}`", the `X` would jump back under the `A` and `E` versus the `C` and `G`, respectively.

```
a:A   B    >C    D    {e:E  F     G}   H    I   <{J  K}   L
⟹
a:A   B     C    D      H   I     L
                      e:E   F     G
            J    K

 c:A  B    >C    (:D  >E    F     G   <<{d:H  I   J}   <K  L)
⟹
 c:A  B     C    (:D   E    F     G)
          d:H    I     J
                       K    L
```

Figure 4. Examples of sub-track specifications. The upper case letters stand for function labels; the lines before the arrow are source text; the two-dimensional arrangement after the arrows shows the intended score positions. (The second example is not a correct C2DA, see the text.)

Hence it appears more regular to inherit no single tab stop (except the own starting score position) from the parent track.

In the functional style, larger segments can be put in parentheses, to make them relative to a preceding or the following label, see section 2.7 below. A sub-track may not start earlier than a containing relative section for a correct indication of the tonic reference point in C2DA. The second example source in Figure 4 cannot change its end "**<K L)**" to "**<<K L)**", because this would move these labels out of the containing parenthesis, which means 'relative to **C**'.

C2DA-subtracking also forbids that a later sub-track is separated from its parent track by an earlier long sibling. The lines in the second example of Figure 4 show the x coordinates of the labels correctly, but are not a correct pen-and-paper rendering: label "**K**" seems to inherit from the track starting with "**d:**", but is meant to inherit from the top-level track starting with "**c:**". Therefore funCode comes with a simple layout algorithm that ensures that every subtrack is optically bound to its parent track, see Figure 5 for a demonstration. This algorithm gets a minimum gap size $g \geq 1$ as its parameter, which ensures that tracks are sufficiently separated when paper lines are reused.

Please note that parsing of C2DA as an input would be even more complicated because you need a 'micro x coordinate' (finer than the mere score position) to distinguish

```
A  (B  C)  D
   E   F
```

from

```
A  (B  C)  D
    E  F
```

2.3 Track Items, Sequential Order and Compound Function

Each track is a sequence of items, see dimension L-2 in Figure 1. The source text of these items will be separated by the funCode parsing process. Most of the items will be function labels, which are assigned to the score positions consecutively.

In most cases, a function label is only one functional symbol, which stands for the simultaneously sounding pitches which realise this function. But for more advanced compositions it is sometimes useful to combine several of these. In C2DA this is simply done by vertical juxtaposition—in funCode by the operator "**&**", see the axis L-3 in Figure 1, the examples in Figures 1 and 10 and the entities **Sum** and **Chord** in Figure 2.

Two further kinds of items are allowed in tracks: "**-**" says explicitly that the preceding item is valid also for this score position, and "**~**" says that there is no entry at all for this particular score position. In a printed or handwritten version, the first is often represented by a similar symbol like '—', the second corresponds to paper left blank. (They are not represented in Figure 2.)

The special *Eureka operator* "**!**" can be prepended to a function label, indicating that the functional sequence up to this point is recognised by the listener only a posteriori when this point is reached. This is a fundamental mechanism in all kinds of *modulations*. In traditional rendering this has been indicated by a backward arrow, see the last track in Figure 6.

2.4 Function Symbols and Their Semantics

The next finer level of the funCode architecture defines the function symbols. Each one consists of a sequence of letters (case is significant, see the non-terminal *rootAndMode* in Table 1), followed by a sequence of numbers, each one possibly decorated (non-terminal *intervalDecorated* in that Table). Each function symbol describes and/or interprets one single chord appearing in the concrete music or abstract pattern under description.

In semantics S-4 (evaluating to pitch classes), the letter sequence merely defines the pitch class of the root of that chord, and whether its third is major or minor.

But in semantics S-1 different character sequences stand for different perceptive situations, different paths in the mental space of harmonic experience. Different character sequences which have identical S-4 semantics stand for different paths which reach the same 'acoustic' end point, see section 1.2 above. For instance "**C:Tps**" and "**C:Sp**" both stand for d minor in semantics S-4. Another example is "**T5_**" to a 'modern' listener, where everybody trained 'classically' hears "**D46+**".

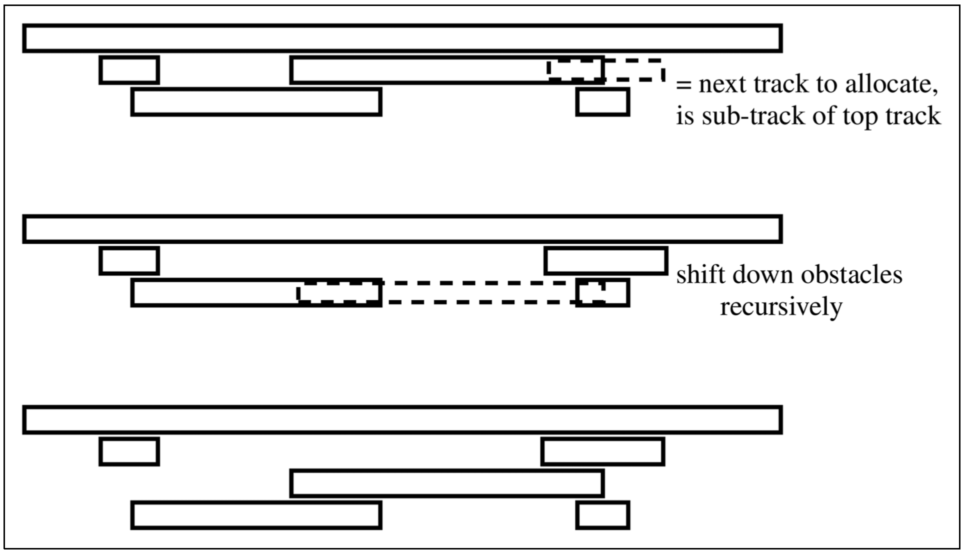
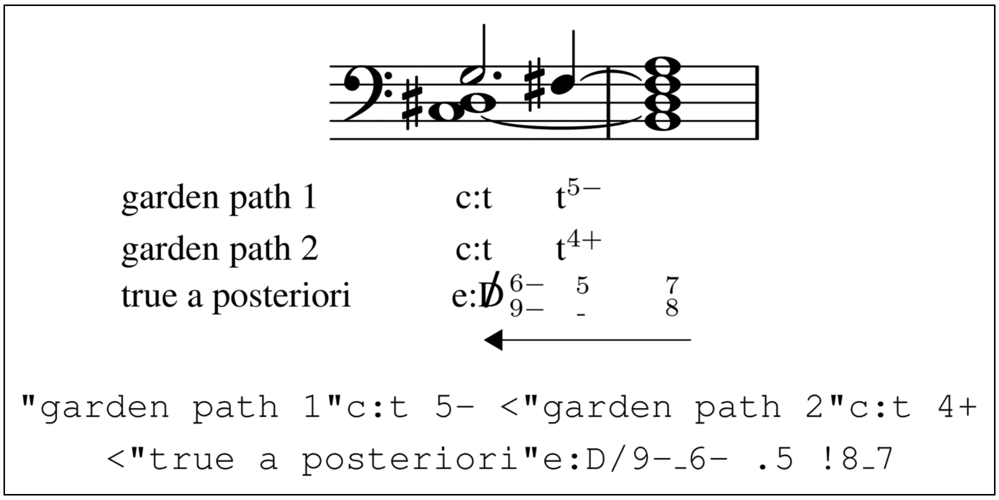


Figure 5. The funCode layout algorithm creates a conventional two-dimensional arrangement (C2DA), in which going upward from the head of each track first hits its parent track. The algorithm visits the tree of sub-tracks in the depth-first manner, siblings sorted by ascending start positions. The dashed box in the first graphic is the third sub-track of the topmost track. Its preceding sibling has already been allocated and is overlapping, thus must be shifted down. The next graphic shows the recursive application of this process, which may affect more than one track.



```
"garden path 1"c:t 5- <"garden path 2"c:t 4+
   <"true a posteriori"e:D/9-_6- .5 !8_7
```

Figure 6. Analysis of the ‘Schicksalsfrage-Motiv’ from ‘Die Walküre’. Above: simplified staff notation of the music; middle: traditional two-dimensional arrangement of labels, including the backward arrow; last two lines: the same in funCode encoding.

Following these letters, a sequence of instances of *intervalDecorated* describes additional intervals added to the triad.

2.5 The Euler Net as the Semantic Domain for Roots and Intervals

As semantic domain for both chord roots and chord components funCode employs the *Euler net* (Tonnetz). This is a two-dimensional vector space of pitch or interval classes, i.e. pitches or intervals modulo octave. The first coordinate represents the exponent with which the interval of the pure fifth is applied, the second that of a pure major third. As usual, a point in this space is identified by a pair of integer numbers (q, t). Any triad thus corresponds to two neighbouring points on a parallel to the axis of fifths (q_0, t_0) and $(q_0 + 1, t_0)$, plus a point $(q_0, t_0 + 1)$ or $(q_0 + 1, t_0 - 1)$ (which is a major third above the lower or below the upper point) for a major or minor triad, respectively. A minor third is represented by the vector $(1, -1)$.

First proposed by Euler (1774), it has since been used in very different variants of music theory. Please note that very different semantics can be assigned: originally it was used to explore the relationship between notation and tuning, and the axes were meant to represent the two intervals verbatim as pure frequency relations. Conversely, the functional theory uses the points to represent psychological situations, inner models or 'situative feelings' of the receiving mind: functional theory implies that the human reception of harmonic processes operates *as if* pure intervals ruled. Even when equal-tempered tuning is applied to produce the sounding physical frequencies of music performance, as it is mostly the case nowadays, the resulting mental representations are different when a process goes two fifths up (from `T` to `DD`) versus to the subdominant's relative minor and then up again (to `SpD`). This is represented *symbolically* by the difference of a syntonic comma, the vector $(4, -1)$ in the Euler net.[11]

The Euler net can be used to represent pitch classes or intervals; both versions are easily convertible but should be distinguished.

Furthermore different notions of enharmonic identity can be applied: originally and in funCode there is none—the vector space is infinite in every direction. When used to model the equal-distance twelve tones, then a cyclic closure is defined after 12 steps on the fifth axis; more recent theories add a cycle after three steps on the third axis, etc.

The Euler net is more expressive than conventional notation, which identifies the pitch classes "`d`" of the fifth of the dominant in C major and the root "`d`", of the relative chord of its subdominant (in funCode notation: `C:D//5` and `C:Sp//1`). Even less expressive are the equal-distance pitch classes (MIDI keys) which identify even A♭ and G♯.

Figure 7 shows some typical variants of a graphical representation of the Euler net: the left version has orthogonal axes for the coordinates of the fifths and the major thirds, thus clearly showing the underlying construction. This form is especially useful when a third orthogonal axis for the pure seventh is added, see Vogel (1975). Its labelling uses one of the many variants to indicate the syntonic comma: the `E,` reached from `C` by a natural third 'is lower' that the `E` reached by four-fifths, which is again lower than the `E'` reached by going one third down and eight fifths up, etc.[12]

Nowadays the variant on the right side of the Figure is often preferred: its didactic advantage is that every set of labels that looks like a minimal triangle indeed represents a triad, which is not the case in orthogonal variant. On the other hand, major and minor thirds seem to be on equal terms, which contradicts the construction. (For a survey on literature about the Euler net see Cohn (2012, pg. 28, 29); see also Cohn (2011) and Gollin (2011).)

2.6 Labelling Root and Mode of Chords

The sequence of characters encoding the root pitch class and the major-minor-mode of the chord thus is interpreted as a vector in the Euler space. Applying this vector to the *current tonic reference point*, as defined by the context of the expression, yields the root pitch. Function $[\![\_]\!]_D$ in Table 2 shows the mapping of the starting points (given by the *tonicCenter* at the start of the track) into the Euler net. Setting `c = (0, 0)` is totally arbitrary; assuming that the key values stand for the 'circle of fifth' can later become critical, see section 2.11 below.

funCode re-models the most widely used system in continental Europe's functional analysis. While historically there were fundamentally different alternatives (Marschner (1894), Oettingen (1913), Keller (1957), Erpf ([1927] 1969), Karg-Elert (1931)), this system (in many slight variations) is the only one which survived and is the pre-dominant in German language literature and many other countries of the European continent. It is based on the works of Grabner (1923) and Maler (1931)[13] and has been further developed and modified by many theorists—for a survey see Imig (1970). Because of those alternatives, it should be called 'GM-style notation' or similar, but in the following, we simply write 'functional'. (funCode itself only adds a few canonical continuations, for instance to allow the operators `D` and `S` in a 'free compositional way' at any position in the string.)

Each instance of *rootAndMode* starts with S, T, or D, followed by arbitrary many P, G, S or D, all in upper or lower case. The upper and lower case indicate major and minor third, so the mode of the constructed triads is immediately visible for the reader. The vectors in the Euler net which correspond to these operator characters are depicted in the left part of Figure 8.

While this calculus indeed is somehow similar to Neo-Riemannian achievements (see section 2.8 below), the main difference is that professionally educated 'classical' musicians in most parts of Europe carry in mind concrete experiences and mental and emotional associations with these symbols (not only semantics S-1 but also S-4, 'intuited chains of intuitions', see above). The remark of Clark (2011, pg. 297) 'classical harmonic thinking [implies] Roman numerals' is simply wrong.

The letter G stands for the German 'Gegenparallelklang', as introduced by Grabner (1923), or simply 'Gegenklang', which corresponds to the English 'counter parallel', and to the 'Leittonwechselklang' of Riemann. The letter P stands for the German 'Parallele', which corresponds to the English 'relative chord'. The application of these operations per se always goes from a minor to a major triad or vice versa. If such a character follows a character of the same case, a change of the third is meant to be appended as a further operation. Writing a change from minor to major as "$\Uparrow$" and the other as "$\Downarrow$", the transformations in the top of Table 3 make these implicit operations explicit.

After these transformations have been applied as often as possible (a process which always terminates) a simple addition of the vector values given by the function $[\![\_]\!]$ in the same Table delivers a vector. Applying this vector to the current tonic reference point delivers the root of the chord. The mode of the chord is simply indicated by the case of the very last character ($\Uparrow$ counting for upper case and $\Downarrow$ for lower.)

With the current tonic reference point set to c, the `Sp` is a d minor chord, `SP` a D major, `SPg` an f sharp minor and

SPG an f sharp major. **SpD** is an A major, namely the dominant to the parallel to the subdominant. Please note that preceding a **d**, **D**, **s** or **S**, a change of mode is redundant because the third is not at all relevant to construct the next chord. Therefore **SPD** is rejected as an error by the reduction in the right part of Table 3.

2.7 Relative Sections

The Euler vector defined by an instance of *rootAndMode* is applied to the *current tonic reference point* to get the root of the chord, all again encoded as coordinates in the Euler net. Every track either declares a *tonicCenter* at its beginning, or it inherits it from its parent track.

But sequences of labels can also be included in parentheses. This means that the tonic reference of all these labels is not the track-wide value but the root of the immediately preceding or following label (= node type **RelativeSection** in Figure 2). Figure 9 schematically shows some examples. The square brackets can be used to notate a chord that does not sound, i.e. which has no correspondence in the concrete music, but nevertheless serves as a reference point (= node type **Virtual** in Figure 2).[14]

Please note that the relative sections need not be properly nested for establishing the tonic reference, see lines f) to i) in the Figure. This is especially convenient for sequences of falling fifth, etc.[15] Relative sections are used in practice quite frequently. They correspond to the 'slash notation' like "**V/V**" from scale degree labelling.

Of course, the track definitions (layer L-1) and relative sections (layer L-2) are designed for 'full compositionality' in the computer science meaning of the term, i.e. for being independently combinable. But some nasty cases of 'feature interaction' have to be treated by the implementation because sub-tracks should not extend beyond a containing relative section. See the technical report Lepper et al. (2022) for details and examples.

2.8 Relation to Neo-Riemannian Triad Transformations

The European GM-style of labelling and the more recent traditions of transformation theory and Neo-Riemannism have the same ancestors and thus much in common. (See Lewin ([1987]2007) and Hyer (1989) for the earliest works and Gollin & Rehding (2011) for a recent survey.) In a major publication from this field, Cohn (2012, pg. 2*ff*) describes as an initial motivating example verbosely the reception process of the beginning of the recapitulation section in the first movement of Schubert's B♭ major piano sonata D 960, measures 217–256. Applying the functional labelling (as taught in many European continental high schools) in a mere schematic way, would end up in measure 256 with the label **Bb:tgPsG** for the finally reached B♭ major. Vice versa, this label can be decoded by every pupil quite fluently as '"B♭ minor – G♭ minor, writable as F♯ minor – A major – d minor – B♭ major'. The one enharmonic exchange is only for ease of spelling—it indicates that the final B♭ major is a priori something different than the starting point. A second and closer look shows that we went three major thirds down (twice a step *x***G** and once a step *x***P**, together with the step **s**), so we are a small diesis higher than we started.

Provocatively we could say that the Neo-Riemannian theorists had to re-discover what is evident to every high school student who grew up with functional instead of scale degree labelling. (*Fortunately* they had to, because underways they discovered a plethora of new viewpoints, analytical tools and modes of presentation.) This is partly admitted by Cohn (2012, pg. xiii): 'Although these [functional] labels are descriptively useful, they do not in

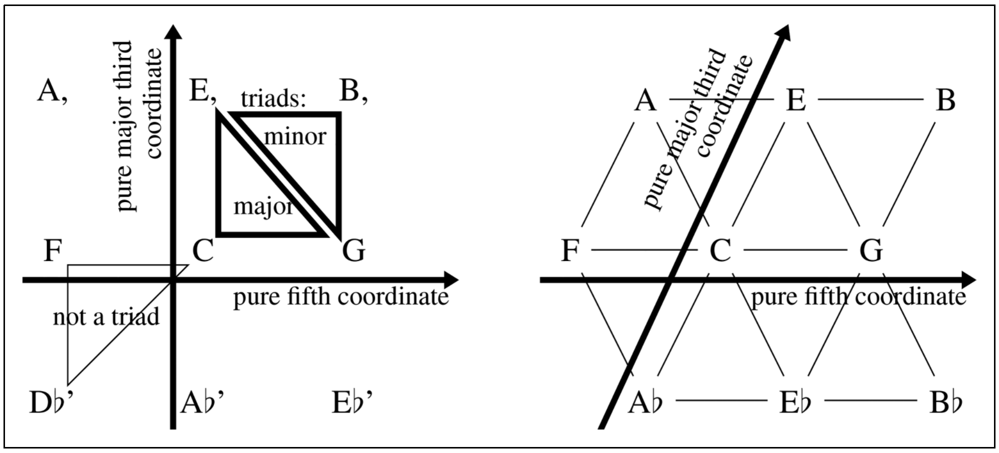


Figure 7. An Euler net is a two-dimensional vector space with discrete points, addressed by integer coordinates. The two axes represent the exponents of the intervals 'pure fifth' and 'pure major third'. Their multiplication originally gave a concrete frequency, but nowadays can also represent a pitch class (in very different tuning systems) or even an affective state in an abstract harmonic space. The left variant with perpendicular axes is the older and more convenient when a third axis shall be added; the right variant has the didactic advantage that all minimal triangles represent musical triads.

themselves lead to an understanding of triadic syntax […] I view these labels as a bridge to a first approximation; what lies on the other side of the bridge […] is an understanding of how the moves designated by these labels behave as part of a compositional system.'

Indeed the functional labels can *directly* be translated into Cohn's operations as shown in the right half of Figure 8.[16] See the Schubert result from above:

`Bb:tgPsG`	⇒ make all mode changes explicit (make **g**, **G**, **p** and **P** always change mode, but **s**, **S**, **d** and **D** never.)
`Bb:tG⇓PS⇓G`	⇒ replace *X*S by *X*pG, *x*s by *x*Gp, *X*D by *X*gP, *x*d by *x*Pg ⇒
`Bb:tG⇓PpG⇓G`	⇒ eliminate adjacent inverses ⇒
`Bb:tG⇓G⇓G`	⇒ translate character symbols from German to English ⇒
`Bb:tLpLpL`	⇒ write both directions of transformation in a unified way ⇒
`Bb:tLPLPL`	

The current implementation of funCode provides automated translation of functional root expressions into L/P/R style coordinates (Lepper et al., 2022). The only strange character remaining in the last line is the "`t`" which says: 'start the transformation chain with the tonic triad in minor mode'. It is not evident why Cohn's L/P/R calculus should be more expressive than the original functional encoding. For instance, the first line shows more clearly that the step from A major to d minor is perceived as a cadential one (`tgP` to `tgPs`) and not as a compound `RLP` step.[17] On the other hand, the last line shows more clearly that we went three major thirds down. And, of course, the much more regular structure of the L/P/R transformations (and their further derivations N/H/S, see Cohn (2012)) connect them much more directly to the mathematical devices like groups, lattices and relations.

2.9 Layer L-4: Chord Components and Voice Leading

The lowest level of the funCode architecture describes the sounding *components of a chord* by giving the interval from its root pitch.

With no explicit interval specification appended, the semantics S-4 realise the tonal function indicated by the function symbol as a concrete triad, with root tone, third and fifth. Further intervals can be added to that chord by appending numbers (with modifiers) to the symbol. The root tone can be explicitly suppressed by a "`/`" character immediately following the root symbol, the other default components by appending `3/` and `5/`. Our canonical continuation of this traditional way of writing are the double slashes as in `D//` which cancel *all* default components of the triad. This allows easy notation of the concept of 'Klangvertretung' (sound substitution), as proposed by Riemann (1882, pg. 185): `D//` is equivalent to `D/3/5/`, and Figure 10 shows two realistic applications.

As a by-product we get a notation for single-pitch chords, which can also be (ab-)used as a notation for a single chord component, like `D//7`.

The syntax and semantics for the additional chord components are configurable, in particular, those for *iModifier*, see section 2.10 for details. The examples used in this article show the variant preferred by its first author.

The top of Table 6 shows the data type of the configuration object, followed by some sample instances. The table at the bottom is a sample declaration of all supported combinations of interval numbers and modifiers.

Some added intervals suppress their default sounding neighbours implicitly; this is configured by the field *suppressDefault* of the configuration object.

Table 4 shows which components of the chords on the different steps of a fixed scale are affected when this scale changes between major and minor: the third, sixth and seventh step of the scale changes, and these pitch classes take descending interval roles when building chords on ascending steps, represented by the descending lines of boxes in the Table. Whenever the root of the chord is affected itself, all pitches *not* affected reflect the major/minor change and vice versa. This leads to a pattern which may be interesting to theoretical reflection but not suited for easy encoding. Indeed, it violates design principle D-2.

So we decided that only the third of each chord is determined by its mode. All further added intervals must be given their size *explicitly*, by an instance of *iModifier*. Therefore the data type of the 'current tonic

Table 2. Normalisation of functions codes and their Euler net semantics.

$$
\begin{array}{llll}
X \in \{\mathtt{T},\mathtt{S},\mathtt{D},\mathtt{G},\mathtt{P},\Uparrow\} & x \in \{\mathtt{t},\mathtt{s},\mathtt{d},\mathtt{g},\mathtt{p},\Downarrow\} & & \\
\ldots X\mathtt{G}\ldots \rightsquigarrow \ldots X\mathtt{g}\Uparrow\ldots & & & \\
\ldots X\mathtt{P}\ldots \rightsquigarrow \ldots X\mathtt{p}\Uparrow\ldots & & \ldots\begin{Bmatrix}\Uparrow\\ \Downarrow\end{Bmatrix}\begin{Bmatrix}\mathtt{S}\\ \mathtt{s}\\ \mathtt{D}\\ \mathtt{d}\end{Bmatrix}\ldots \rightsquigarrow \mathtt{error}(\text{“\textit{Superfluous}”}) & \\
\ldots x\mathtt{g}\ldots \rightsquigarrow \ldots x\mathtt{G}\Downarrow\ldots & & & \\
\ldots x\mathtt{p}\ldots \rightsquigarrow \ldots x\mathtt{P}\Downarrow\ldots & & & \\
[\![\mathtt{g}]\!] = (0,1) & [\![\mathtt{G}]\!] = (0,-1) & [\![\mathtt{p}]\!] = (-1,1) & [\![\mathtt{P}]\!] = (1,-1) \\
[\![\mathtt{T}]\!] = [\![\mathtt{t}]\!] = [\![\Uparrow]\!] = [\![\Downarrow]\!] = (0,0) & [\![\mathtt{D}]\!] = [\![\mathtt{d}]\!] = (1,0) & [\![\mathtt{S}]\!] = [\![\mathtt{s}]\!] = (-1,0) &
\end{array}
$$

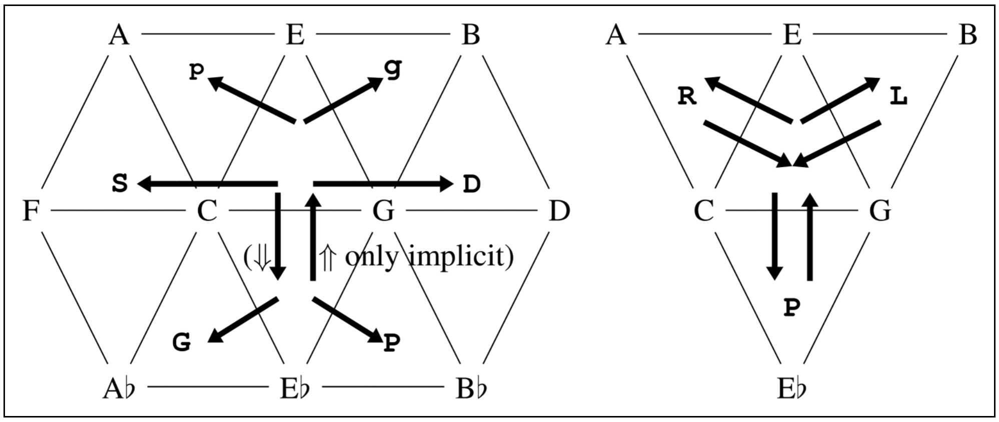


Figure 8. Relations between triads, according to Grabner (1923)/Maler (1931) and Cohn (2012).

Table 3. Encoding of the tonic centers, i.e. of the starting points in the Euler net. The origin $[\![\texttt{C}]\!]_{\mathrm{D}} = (0,0)$ is chosen arbitrarily.

$[\![\texttt{F}]\!]_{\mathrm{D}} = (-1,0)$	$[\![\texttt{C}]\!]_{\mathrm{D}} = (0,0)$	$[\![\texttt{G}]\!]_{\mathrm{D}} = (1,0)$	$[\![\texttt{D}]\!]_{\mathrm{D}} = (2,0)$
$[\![\texttt{A}]\!]_{\mathrm{D}} = (3,0)$	$[\![\texttt{E}]\!]_{\mathrm{D}} = (4,0)$	$[\![\texttt{B}]\!]_{\mathrm{D}} = (5,0)$	
$[\![x\texttt{\#}]\!]_{\mathrm{D}} = [\![x]\!]_{\mathrm{D}} + (7,0)$		$[\![x\texttt{b}]\!]_{\mathrm{D}} = [\![x]\!]_{\mathrm{D}} - (7,0)$	
$[\![x\texttt{,}]\!]_{\mathrm{D}} = [\![x]\!]_{\mathrm{D}} - (4,-1)$		$[\![x\texttt{'}]\!]_{\mathrm{D}} = [\![x]\!]_{\mathrm{D}} + (4,-1)$	

a)	`A (B C D) E`	Roots of B, C and D are defined relative to the root of E
b)	`A (B C D:) E`	Same as line a)
a)	`A (B C D) [E]`	Same as line a), but E does not sound = does not correspond to a score event
c)	`A (:B C D) E`	Roots of B, C and D are defined relative to the root of A
d)	`A ((B C) D) E`	Roots of B and C are relative to the root of D, which is relative to E
e)	`A (B (:C D)) E`	Roots of C and D are relative to the root of B, which is relative to E
f)	`A (((B) C) D) E`	Root of B is relative to C is relative to D is relative to E
g)	`A ((B) C) (D) E`	Same as line f)
h)	`A (B) (C) (D) E`	Same as line f)
i)	`A (B) ((C) D) E`	Same as line f)

Figure 9. Relative sections. The upper case letters stand for function labels.

reference' of each track is indeed only a pitch class, without any mode indication. As a consequence, tonal key does *emerge* from harmonies but is never given a priori, as such. This corresponds to the more modern ways of composition, starting with late Schumann, Liszt, and Bruckner, and to the notion of tonality as defined by Schenker (1906), which is always a mixture ('Mischung') of major and minor steps. As a consequence, only the size of the third is indicated by the (upper or lower case) of the functional symbol, all other intervals (suspensions, transitions or additions) must be qualified explicitly.[18]

The configuration from Table 6 defines the modifiers **-**, ε, **+** and **++** for the 'perfect' intervals 1, 4, 5 and 8, meaning diminished, perfect, augmented and double-augmented respectively, and **--**, **-**, **+**, and **++** for the 'imperfect' intervals 2, 3, 6, 7, 9, meaning diminished, small, large, and augmented. But very different notation systems and sets of allowed interval sizes are frequently used and can be plugged in by redefining the map in *intervals*.

For mere convenience, a replacement text for a **7** without modifier over a root symbol ending with **D** can be defined, which allows to write **D7** or **D79-** instead of **D7-** and **D7-9-**, etc.

Describing the components of a chord by combining numbers and modifiers had been invented by Riemann (1880) and called 'Klangschlüssel' (= 'sound key'). First, he applied it not to function symbols but to names of

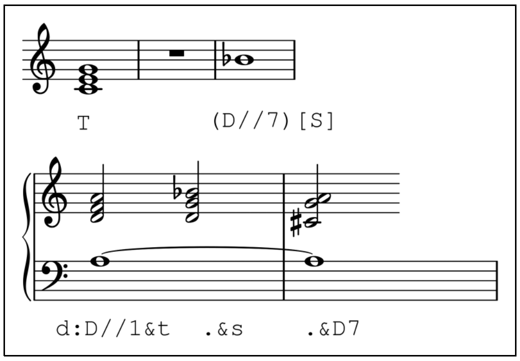


Figure 10. Labelling the concepts of 'Klangvertretung' (sound substitution) and organ point.

concrete pitch classes, as it is nowadays widely done in 'lead sheet' notation in popular music. To eliminate any ambiguity, tone names may be followed by a colon character in the context of funCode. So `D:7-9-` means a D-major-seven-nine chord, but `D7-9-` means a dominant relative to the current tonic centre.

But also in pure theory, a compact and non-ambiguous notation of interval structure is desirable; see Table 5 for several examples of complex chord structures with very idiosyncratic names in German, taken from a standard text book on scale degree theory by Ganter (1975), and for the names of the chords of the augmented sixth in English theory.

The number stacks seem to be borrowed from the Baroque 'figured bass' notation, but indeed mean something fundamentally different, namely intervals relative to the root note, not to the bass note. The bass note and possibly the melody tone can be indicated by appending an underscore "_" or the caret "^" to the interval specification.[19] An error is signalled by funCode if one of these signs is present more than once.

Another relic from figured bass is that the number stack is 'abused' to indicate voice leading: if the *rootAndMode* is verbatim the same as its predecessor (not regarding the case of the very last character) or even totally absent, then the number stack is related to the preceding number stack. Dots "`.`" in the stack stand for unaltered stack positions (in the traditional rendering often printed or hand-written as '–'), i.e. for the same tone in the preceding and in the current chord, while numbers replacing another number are *possibly* meant as 'voice leading': the c going to the b in Figure 1 is indicated by the **4** replaced by the **3**. But that is not necessarily so. When applying semantics S-4 it merely means that some pitch class is no longer present but replaced by some other, saying nothing about octaves and voice leading.

2.10 Localisation and Customisation

The evaluation process of chord bases and intervals is more or less controlled by a configuration object, see the data type *Configuration* in Table 6. It supports far-ranging customisation and localisation. This is important for the acceptance and practical usability of funCode, because a theorist might be used to a particular idiom which they want to keep as far as possible when switching to a digital and more formalised representation, see design principle D-4 in section 1.4.

funRenamings is applied to the input conceptually before the evaluation described by Table 3 takes place. This allows localisation of function symbols, like '`C`' or even '`Cp`' for 'counter-parallel' instead of '`G`' for German 'Gegenklang'.

Similar, *keyRenamings* is applied before $[\![-]\!]_D$ from Table 2 is evaluated, allowing national pitch names like 'la' or 'h'.

chordAbbreviations are added as further alternatives to the definition of the non-terminal *rootAndMode* in Table 1 and expanded according to their definition. These short-cut rules are not important for machine processing, but for readability and writability by humans.

The configuration object $\mathcal{C}_{\text{conv}}$ in Table 6 contains two widely used abbreviations for the chord of the diminished seventh and the chord of the Neapolitan sixth. Please note that these abbreviations are expanded verbatim: they contain interval digits, so they put an end to the containing character sequence.

allowed:	`(sN`	`D)`	`D`	`T`
allowed:	`DsN`	`DD`	`D`	`T`
both expand to:	`DsG3_`	`DD`	`D`	`T`
allowed:	`(D)`	`sN`	`D`	`T`
not allowed:	`sND`	`sN`	`D`	`T`
because it expands to:	`sG3_D`	`sG3_`	`D`	`T`

The customisation mechanism can be employed for more precision. In English literature we found for instance the abbreviation "`Gr`" used in different articles with different meaning, see the last three lines in Table 5: most authors restrict it to `DD/5-_79-`, some also include `D/5-_79-`, and Heetderks (2015) uses it in his Figure 8 for `DD/3_5-79-`. A simple but fully formal declaration at the begin of an article (like "`Gr:=DD/5-79-`") would clarify and allow the well-readable notation "`Gr3_`".

The further fields in the data type *Configuration* in have already been described above: the set of all intervals, which make up the concrete chord at the given score position, is calculated from the explicitly mentioned intervals and the defaults. The default mechanism is again not required by machine processing but for convenient reading and writing by humans.

The map *intervals* gives the external representations for the corresponding Euler values. (In Table 6 the example *stdIntervals* is printed as a two-dimensional table, but this means indeed a map of type String ⇸ CEuler.)

All intervals in the set *defaults* are added to the the set of pitch classes, unless the interval's number (stripped from the modifier) appears in the explicit intervals, or a suppressing interval appears there, as defined by the map *suppressDefault*. So in standard 'classic' usage of harmony, the third and the fifth are in every chord, but not if a

Table 4. Chord components affected by the global mode of the containing scale. The boxes show the scale degrees affected by a change from major to parallel minor (here: c) and their roles in the chords on all degrees of the scale. The numbers show the chord components affected by that change.

b/b♭=	$\boxed{7}$		7	$\boxed{7}$		7	$\boxed{}$
a/a♭=	$\boxed{6}$	$\boxed{6}$	6		$\boxed{6}$	6	6
g		$\boxed{5}$	$\boxed{}$			$\boxed{}$	5
f			$\boxed{}$	$\boxed{4}$		4	$\boxed{}$
e/e♭=	$\boxed{3}$		3	$\boxed{3}$	$\boxed{3}$	3	3
d		$\boxed{2}$	2		$\boxed{2}$	$\boxed{}$	2
c			$\boxed{}$			$\boxed{}$	$\boxed{}$
I		II	III	IV	V	VI	VII

$V = \{3, 6, 7\}$

$affected(r, i) = [(r \in V) \neq ((r + i - 1)\text{mod}7 \in V)]$

'suspension' is signalled in the intervals. Consequently **4** implies **3/** and **6-** and **6+** imply **5/**. E.g. in the hypothetical example $\mathcal{C}_{\text{blues}}$ in Table 6, all chords contain a minor seventh by default and an explicit **6+** implies **7/**.

The flexibility of interval encoding is necessary to cover different functional theories. For instance, there exist numerous different opinions about the (physical and psychological) nature of the seventh in the dominant function.[20] By defining for instance "$\mathbf{7+} \mapsto \text{CEuler}(-2, 0), \mathbf{7\star} \mapsto \text{CEuler}(2, -1), \mathbf{7v} \mapsto \text{CVogel}(0, 0, 1)$", three different variants can even coexist and be applied occasionally (as soon as *CEuler* has been embedded into the three-dimensional vector space *CVogel* to support the natural seventh).

2.11 Enharmonic Adjustments

Up to here, the semantics of one single annotation track (more precise: its S-4 semantics) have been calculated relative to an arbitrarily chosen starting point. As soon as two such tracks overlap, which are intended to be simultaneously valid (i.e. the T-0 use case from above), the Euler coordinates should be adjusted accordingly, to reflect their mutual relation.

The upper part of Figure 11 shows an example of the famous 'syntonic mill': simply evaluating the chord roots according to the annotation delivers the coordinates in large font. But since the first overlap in both tracks shows a difference of the syntonic comma (4,-1), the following coordinates should be adjusted to the sequence shown in the box, which does not return to the starting point.

The lower part shows the well-known text-book enharmonic modulation from c major to d♭ major via the 'German Sixth'. It shows that simply comparing the chord roots is not sufficient but must be extended to all notes. In this context, all kinds of *enharmonic steps* must be considered, like (12, 0), (4, 2), (0, 3), (4, − 1) etc. For this, an algorithm has not yet been implemented.

3 Conclusion

Mathematical remodelling of existing cultural symbol systems aims at clarifying syntax and semantics for both human discourse and automated processing. It forces all conventional implicit assumptions to be discussed explicitly and encourages us to straighten out rough edges of historical origin, often by canonical continuations.

funCode applies mathematical remodelling to a widely used system of functional harmonic labelling (Grabner–Maler style = GM-style). It is intended as a target format for the transcription of existing analyses (e.g. for automated processing) and as a versatile and adaptive labelling system on its own.

In course of the analysis of existing practice, it first turned out that fundamentally different kinds of the semantics of harmonic labelling per se must be distinguished (section 1.2).

Transforming current practice into a mathematical model yields four more or less independent strata. Only one of them is specific to functional theory (see Figure 1 and section 1.5).

Table 5. Examples for interval specifications and their idiosyncratic namings.

"Vermindert-verminderter Septakkord"	`D/79-`
"Dreifach verminderter Septakkord"	`D/5-79-`
"klein-doppeltvermindert-verminderter Septakkord"	`D/7--9-`
"groß-vermindert-kleiner Septakkord"	`D5-7`
"übermäßig-kleiner Septakkord"	`D5+7`
"zweifach vermindert-kleiner Septakkord"	`D/5-79+`
"vermindert-doppeltvermindert-verminderter Septakkord"	`D/5-7--9-`
"groß-vermindert-verminderter Septakkord"	`D5-7--`
"Septnonenakkord mit hoch- und tiefalterierter Quinte"	`D5-5+79+`
"French Sixth"	`DD5-_7`
"Italian Sixth"	`DD/5-_7`
"German Sixth"	`DD/5-_79-`
idem	`D/5-_79-`
idem	`D/5-79-`

Table 6. Configuration and localisation of funCode.

Configuration

$$
\begin{array}{l}
\mathit{keyRenamings} : \texttt{String} \nrightarrow \texttt{String}\\
\mathit{funRenamings} : \texttt{String} \nrightarrow \texttt{String}\\
\mathit{chordAbbreviations} : \texttt{String} \nrightarrow \texttt{String}\\
\mathit{intervals} : \texttt{String} \nrightarrow \textsf{CEuler}\\
\mathit{abbreviateD7} : \textsf{opt}\ \texttt{String}\\
\mathit{defaults} : \mathbb{P}\mathcal{L}(\mathit{intervalDecorated})\\
\mathit{suppressDefault} : \mathcal{L}(\mathit{intervalDecorated}) \leftrightarrow \mathcal{L}(\mathit{intervalDecorated})
\end{array}
$$

$$
\begin{array}{ll}
\mathcal{C}_{\text{pure}} = \langle & \mathit{keyRenamings} \Rrightarrow \emptyset;\\
& \mathit{funRenamings} \Rrightarrow \emptyset;\\
& \mathit{chordAbbreviations} \Rrightarrow \emptyset;\\
& \mathit{intervals} = \textsf{stdIntervals};\\
& \mathit{abbreviateD7} \Rrightarrow \bot;\\
& \mathit{defaults} \Rrightarrow \{\text{“}\mathbf{1}\text{”}, \text{“}\mathbf{3}\text{”}, \text{“}\mathbf{5}\text{”}\}\\
& \mathit{suppressDefault} = \{\mathbf{2+} \mapsto \mathbf{1}, \mathbf{2-} \mapsto \mathbf{1}, \mathbf{4} \mapsto \mathbf{3}, \mathbf{2++} \mapsto \mathbf{3},\\
& \qquad \mathbf{6+} \mapsto \mathbf{5}, \mathbf{6-} \mapsto \mathbf{5}\}\\
& \rangle
\end{array}
$$

$$
\begin{array}{l}
\mathcal{C}_{\text{blues}} = \mathcal{C}_{\text{pure}}\langle \mathit{defaults} \Rrightarrow \{\text{“}\mathbf{1}\text{”}, \text{“}\mathbf{3}\text{”}, \text{“}\mathbf{5}\text{”}, \text{“}\mathbf{7-}\text{”}\}\\
\qquad\qquad \mathit{suppressDefault} \Rrightarrow (\ldots \mathbin{⩥} \{\mathbf{5}\}) \cup \{\mathbf{6-} \mapsto \mathbf{7-}, \mathbf{6+} \mapsto \mathbf{7-}\}\rangle\\
\mathcal{C}_{\text{conv}} = \mathcal{C}_{\text{pure}}\langle \mathit{chordAbbreviations} \Rrightarrow \{\mathbf{sN} \mapsto \mathbf{sG3}_{-}, \mathbf{DV} \mapsto \mathbf{D/79-}\};\\
\qquad\qquad \mathit{abbreviateD7} \Rrightarrow \mathbf{7-}\rangle\\
\mathcal{C}_{\text{DE}} = \mathcal{C}_{\text{conv}}\langle \mathit{keyRenamings} \Rrightarrow \{\mathbf{H} \mapsto \mathbf{B}, \mathbf{B} \mapsto \mathbf{Bb}\}\rangle\\
\mathcal{C}_{\text{DE2}} = \mathcal{C}_{\text{DE}}\langle \mathit{keyRenamings} \Rrightarrow \ldots \cup \{\mathbf{As} \mapsto \mathbf{Ab}, \mathbf{Ais} \mapsto \mathbf{A\#},\\
\qquad \mathbf{Ces} \mapsto \mathbf{Cb}, \mathbf{Cis} \mapsto \mathbf{C\#}, \mathbf{Des} \mapsto \mathbf{Db}, \mathbf{Dis} \mapsto \mathbf{D\#}, \mathbf{Es} \mapsto \mathbf{Eb},\\
\qquad \mathbf{Fis} \mapsto \mathbf{F\#}, \mathbf{Ges} \mapsto \mathbf{Gb}, \mathbf{Gis} \mapsto \mathbf{G\#}\}\rangle\\
\mathcal{C}_{\text{UK}} = \mathcal{C}_{\text{conv}}\langle \mathit{funRenamings} \Rrightarrow \{\mathbf{Cp} \mapsto \mathbf{G}\}\rangle\\
\mathcal{C}_{\text{Cohn}} = \mathcal{C}_{\text{conv}}\langle \mathit{funRenamings} \Rrightarrow \{\mathbf{L} \mapsto \mathbf{G}, \mathbf{R} \mapsto \mathbf{P}\}\rangle
\end{array}
$$

stdIntervals =

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
+		$(2,0)$	$(0,1)$	$(2,1)$	$(0,2)$	$(-1,1)$	$(1,1)$		$(2,0)$
++		$(1,2)$				$(2,2)$			
−		$(-1,-1)$	$(1,-1)$		$(-2,-1)$	$(0. -1)$	$(-2,0)$		$(-1,-1)$
ϵ	$(0,0)$		(dep.)	$(-1,0)$	$(1,0)$		D : **7−**	$(0,0)$	

The complete grammar for all four strata is specified in the 18 lines of text in Table 1. The complete semantics are specified as an 'executable meta-model' by some 800 lines of Prolog code in the accompanying technical report (Lepper et al., 2022).

On the top level L-1, the C2DA of tracks and labels is represented by a linear encoding using tab stops and braces (section 2.1). In spite of its seeming simplicity, numerous subtle pitfalls have been identified, concerning possible ambiguities of inheritance relations and the nesting of tracks and relative sections (section 2.2, Figure 5 and the technical report).

The next lower level L-2 is the horizontal sequence of items in one single track, each item related to one abstract score position. On this level the rules for relative references have been formalised (section 2.7, Figure 9).

Level L-3, the combination of more than one functional chord into one label, turned out as the least problematic.

The lowest level L-4 (covering functional symbols, root pitch classes and chord components) employs the Euler net as its semantic domain (section 2.5, Figures 7 and 8). A close relation of GM-style and Neo-Riemannian triad transformations is obvious which is discussed in section 2.8 and is supported by additional translation functions in the implementation.

To be acceptable by practitioners, all levels of the system's architecture must be adaptive to the different usus of schools and even to personal styles of authors: different abbreviations of conventional pitch classes and of function symbols can be plugged into the system. Especially the modifiers for interval modes (major, minor, diminished, augmented, etc.) can be defined freely, together with the sets of both default intervals and suspension rules. This can make funCode useful also in 'non-classical' contexts (section 2.10, Table 6).

A Transcriptions of Neo-Riemannian Examples

The following lines give without any comment the transcriptions of examples from major publications of the Neo-Riemannian school, to demonstrate their combination of expressive power with concrete connotations in any practical musician's mind. The transcriptions reflect the 'physical' semantics (see section 1.2); alternatives according to S-1 to S-3 are perhaps possible. An *animated presentation* as suggested by Cohn (2011, pg. 330) could easily be

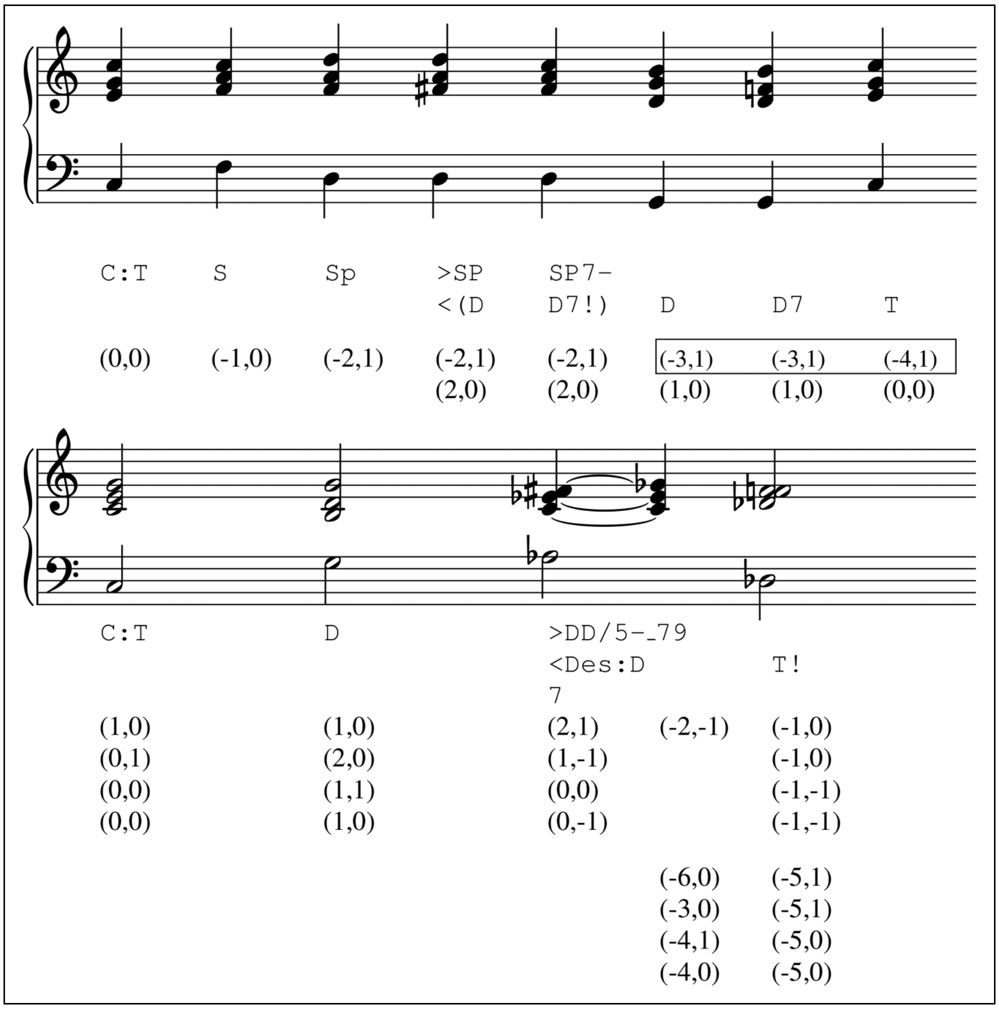


Figure 11. Syntonic deviations.

implemented based on the funCode implementation contained in Lepper et al. (2022).

Diagram ‘Ex. 10.8’ by Clark (2011):
“`g#:t tP tp tpG {tP tG} {tP tpS} (D) Tg`”

Diagram ‘Ex. 10.9’ ibd., with “`|`” standing for the separating bar lines:
“`c:t tP tpG tpgP | T tG s Tp (:D) | (:TP) (:S) (:S) (:S)`”

Diagram ‘Ex. 10.12’ ibd.:
“`Bb:t tP tp tpG tps tpS`”

Engebretsen (2011, pg. 364): “$\mathcal{S}^{3>} - \not{\mathcal{D}}$” becomes “`sG DD`”.

Ibd., pg. 368, Riemann’s Example 85: “`c:T ss s3_ D6+4 7 T`” (The second D♭5 note head seems an engraving error for C5.)

Example 86: “`a:t DD D t`”

Example 87a: “`a:tP D t`”; example 87b: “`c:Tp s T`”

Example 88a: “`c:T tG s56+ D46_53 t`”; 88b: “`c:T tG (D46+ 53) tP`”; 88c: “`a:t Tg D5_7 T`”; 88d: “`c:t Tg (s56+) Tp`”

Example 89a: “`c:T tG46- 53`”; Example 89b: “`a:t (D) Tg`”

Example 90a: “`c:T (D) Sp`”; Example 90b: “`a:t (s) dP`”, etc.

The newly defined mediant operations by Kopp (2011) have been covered by GM-style ever since: ‘USM’ = “`TG`”, ‘UFM’ = “`tP`”, ‘LSM’ = “`TP`”, and ‘LFM’ = “`tG`”.

Acknowledgements

The authors thank all reviewers for their valuable suggestions to improve this article.

Action Editor

David Meredith, Department of Architecture, Design and Media Technology, Aalborg University.

Peer Review

Nick Harley, Vrije Universiteit Brussel, Artificial Intelligence Lab. Maximos Kaliakatsos-Papakostas, Athena Research and Innovation Center in Information Communication and knowledge Technologies.

Declaration of Conflicting Interests

The authors declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The authors received the following financial support for the research, authorship, and/or publication of this article: Open access publication of this article has been financially supported by Deutsche Forschungsgemeinschaft (DFG) and Open Access Publishing Fund of Osnabrück University.

ORCID iD

Markus Lepper https://orcid.org/0000-0002-9120-3908

Notes

1. Learning from mathematics is often sensible: contemporary systems use the wording ‘pitch class’, which is borrowed from the theory of congruence relations in mathematics. For the same thing, older theories had to invent their own, stand-alone nomenclature, like ‘Elementarfeld’ by Keller (1957).
2. There have been some attempts for compromise. For example, Arnold Schoenberg uses roman numeral theory in his Harmonielehre (Schoenberg, 1922), but integrates functional symbols in his later work (Schoenberg, 1954).
3. The formula is given in the widest spread variant of the traditional functional notation (Imig, 1970, pg. 223). Possible funCode encodings of the three examples are **`(D7) sG3_ ! D7 t`**, **`DD5-7 D7 t`** and **`DD7 D7 t`**. (The first term describes an augmented cadence where a dominant is applied to the second chord which is the Neapolitanian subdominant; then the normal dominant seventh and the minor tonic follow. The second and third terms describe a cadence starting with an applied dominant, which has a diminished or pure fifth, respectively.)
4. S-5 is currently not supported by funCode. For this, the semantic sphere needed a more complex data type than just sets of pitch classes, but a notion of voice or octave, or similar. But the current implementation is able to process symbol sequences meant according to S-5 by its author and produce S-4 results: the sequence ‘**`t24 33`**’ in S-5 represents a double suspension, including the correct voice leading. In our implementation, it is a valid input, and its S-4 interpretation is simply two sets of pitch classes, so it is equivalent to ‘**`t24 t`**’.
5. The early and far-going examples for a canonical continuation are the ‘Riemann Systems’ by Lewin (1982), where the ‘triad of triads’, as explored by Hauptmann (1873), is enhanced to arbitrary generator intervals and tested a posteriori for their applicability in musical practice.
6. ‘Aus praktischen Erwägungen ist es w unschenswert, dass sich die Akkordstruktur (Intervallaufbau) unmittelbar aus der Funktions- oder Stufenbezeichnung ablesen läßt.’ (Hussong, 2005, pg. 99) (‘For practical reasons it is desirable that the chord structure (interval structure) is immediately evident from the symbol for the function or scale degree.’) In particular this should be *context free* and *transposition invariant*. Commonly, the function ‘tP’ is written as “♭III” in C major, but as “♮III” in C sharp major, which clearly violates these principles. This is a typical example of the confusion between ‘external representation’ (= note writing = syntactic sphere) and intended semantics, often found in music theory.
7. ‘Eleventh and thirteenth chords, in particular, have a compound nature, consisting of one chord superimposed over another.’ (Hewitt, 2011, pg. 254). So what is labelled in lead sheet notation as ‘11’ or ‘13’ often corresponds to axis L-3.
8. According to Hentschel et al. (2021), always ‘a *segment* [of a score is] referred to by [a] chord symbol’ (our emphasis). In our model and in C2DA this segment extends from the current score position to its successor.
9. A major example for a theoretic approach is the characterisation of three possible listener types when hearing ‘Himmel nimm des Dankes Zähren’ from *Freischütz* by Ploeger (1990, pg. 38).
10. The tracks in Figure 1 stand for different encoding styles, e.g. by different authors, thus show case T-3, while those in Figure 6 show the different effects which simultaneously affect the listener = T-0.
11. Hewitt (2000, pg. 43,44) states ‘[The interpretation that] enharmonically equivalent intervals [..] are the same interval spelled differently [..] fails to take into account the various psycho-acoustical and *psychological* modifications that may be made be the perceiver’ (our emphasis). ‘[E]qual temperament is an unsatisfactory theoretical base from which to approach and understand musical intervals.’ and cites Mitchell (1962) ‘[..] whatever pair of pitches might represent an isolated sound, as soon as it participates in a contiuum of musical relationship, its nature becomes completely dependent on its surroundings.’ See also the detailed discussion by Ploeger (1990, pg. 47*ff*)
12. Since the nodes of the Euler net represent classes of pitches modulo octave, the correct statement is of course ‘taking one representative each, with minimal distance, the representative of **`E,`** is lower than that of **`E`**.’
13. See Holtmeier (2011, pg. 12p, 40*ff*) for a discussion of the different issues of these works and of the involvement of both authors in politics, degenerating their own achievements. We do not agree that ‘it was only its specific development in National Socialism that led to its monopoly, which allowed Maler’s function symbols to reach virtually all institutions of higher education after the Second World War.’ As Holtmeier says himself, the definition of their symbol system took place in their ‘open-minded’ pre-Nazi issues, and later it got accepted for its clarity, compactness, expressiveness and wide applicability, demonstrated therein. ‘Die Einfachheit der Funktionssymbole ist ein wichtiger Faktor, der mit entscheidet, ob sich ein Harmoniesystem durchzusetzen vermag oder nicht’. (‘The simplicity of the functional symbols is a decisive factor for the assertiveness of a harmonic system.’) (Imig, 1970, pg. 229).
14. The corresponding non-terminal in Table 1 is $rootAndMode_{\mathrm{N}}$, which is further restricted not to end with an operation ⇑ or ⇓, see technical report Lepper et al. (2022) for details.
15. Lewin ([1984]2006, pg. 192 writes ‘S(S(S(S(S(S)))))’ for the ‘plagal power of D major’ at the beginning of the six plagal steps at the end of *Parsifal*. In any coding system this is erroneous: conventionally he must write ‘(((((S)S)S)S)S)S’. funCode’s canonical continuation allows ‘(S)(S)(S)(S)(S)S’, saying simply that every chord functions as subdominant for its follower, independent of its definition.

16. These operations were introduced as "**DOM**", "**SUBD**", "**MED**", "**SUBM**", etc. by Lewin ([1987]2007). They were transformed into the operators "**L**", "**R**", "**D**", and "**P**" by Hyer (1989, pg. 162*ff*), written as *postfix* operators and linked to the Euler net as their semantic domain (pg. 190*ff*).
17. The step types **s** and **RLP** have identical semantics according to S-1, but differ according to S-4. See their discussion above. Similar objections by David Kopp (Clark, 2011, pg. 300).
18. The relevance of a global mode indication to the size of additional intervals is always critical. E.g. Kurth in his famous interpretation of the begining of 'Tristan' explicitly mentions the f natural instead of f sharp in the first chord (Hyer, 1989, pg. 3), in spite of a **DD5-** being the default in a minor key since Beethoven's first piano sonata. Even a text book addressing popular musicians supports our viewpoint: "Modal interchange […] led to […] a common pot of chords [which] contains all of the chords from both the major and the minor keys. [Thus] chord progressions […] are difficult to classify as being in the major or the minor key. […] If the tonic triad is major, then the music will sound like it is written in the major key; whereas it is minor, it will sound like it is written in the minor key. This is irrespective of what other chords are being used." (Hewitt, 2011, pg. 236)
19. These roles have been introduced by Georg Capellen, 1903 (Imig, 1970, pg. 143)
20. See among many others Hauptmann (1873, pg. 114), Riemann (1918, pg. 142), Imig (1970, pg. 86), Vogel (1975, pg. 92, referring to Leipniz and Euler), and Hewitt (2000, pg.112) for different standpoints.

References

Broy, M. (2011). Informatik als wissenschaftliche Methode: Zur Rolle der Informatik in Forschung und Anwendung. In: *41. Jahrestagung der Gesellschaft für Informatik*, *LNI*, volume P192. Berlin. ISBN 978-3-88579-286-4, pp. 43–44.

Cambouropoulos, E., M Kaliakatsos-Papakostas., & C Tsougras. (2014). An idiom-independent representation of chords for computational music analysis and generation http://hdl.handle.net/2027/spo.bbp2372.2014.155.

Clark, S. (2011) On the imagination of tone in Schubert's Liedesend (D473), Trost (D523), and Gretchens Bitte (D564). In: Gollin & Rehding (2011).

Cohn, R. (2011). Tonal pitch space and the (neo-)Riemannian Tonnetz. In: Gollin & Rehding (2011).

Cohn, R. (2012). *Audacious euphony — chromaticism and the triad's second nature*. New York, NY: Oxford Press. ISBN 978-0-19-977269-8.

Engebretsen, N. (2011). Neo-riemannian perspectives on the Harmonieschritte. In: Gollin & Rehding (2011).

Erpf, H. ([1927]1969). *Studien zur Hharmonie- und Klangtechnik der neueren Musik*. Wiesbaden: Breitkopf & Härtel.

Euler, L. (1774).De harmoniae veris principiis per speculum musicum repraesentatis. *Novi Commentarii academiae scientiarum Petropolitanae* 18.

Forte, A. (1973). *The structure of atonal music*. New Haven: Yale University Press.

Ganter, C. (1975). *Die dur-moll tonale Harmonik – Teil III: Chromatik*. Basel: Hega-Verlag.

Gollin, E. (2011). From matrix to map: Tonbestimmung, the Tonnetz, and Riemann's combinatorial conception of interval. In: Gollin & Rehding (2011).

Gollin, E., & Rehding, A. (eds.) (2011). *The Oxford manual of neo-riemannian music theories*. New York, NY: Oxford Press. ISBN 978-0-19-522133-3.

Grabner, H. (1923). *Die Funktionentheorie Hugo Riemanns und ihre Bedeutung für die praktische Analyse*. München: Leuckart.

Hauptmann, M. (1873). *Die Natur der Harmonik und Metrik*. Leipzig: Breitkopf und Härtel.

Heetderks, D. (2015). From uncanny to marvelous: Poulenc's hexatonic pole. *Theory Pract*, *40*, 177–204. https://www.jstor.org/stable/10.2307/26477736.

Hentschel, J., Moss, F. C., McLeod, A., Neuwirth, M., & Rohrmeier, M. (2021). Towards a unified model of chords in western harmony. In: *proceedings of the Music Encoding Conference MEC 2021, in preparation*.

Hewitt, M (2000). *The tonal phoenix*. Bonn: Orpheus Verlag.ISBN 3-922626-96-3.

Hewitt, M. (2011). *Harmony for computer musicians*. Boston, MA: Course Technology. ISBN 1-4354-5672-6.

Holtmeier, L. (2011). The reception of Hugo Riemanns music theory. In: Gollin & Rehding (2011).

Hussong, H. (2005). *Untersuchungen zu praktischen Harmonielehren seit 1945*. Berlin: Verlag im Internet GmbH.

Hyer, B. (1989). *Tonal intuitions in Tristan und Isolde*. Ann Arbor, MI: University Microfilms International. ISBN 1-4354-5672-6.

Hyer, B. (2011). What is a function? In: Gollin & Rehding (2011).

Imig, R. (1970). *Systeme der Funktionsbezeichnung in den Harmonielehren seit Hugo Riemann*. Düsseldorf: Gesellschaft zur Förderung der systematischen Musikwissenschaft e.V.

Karg-Elert, S. (1931). *Polaristische Klang- und Tonalitätslehre*. Leipzig: Leuckart.

Keller, W. (1957). *Handbuch der Tonsatzlehre*. Regensburg: Bosse.

Kopp, D. (2011). Chromaticism and the question of tonality. In: Gollin & Rehding (2011).

Lepper, M. (2021). *de linguis musicam notare — Beiträge zur Bestimmung von Semantik und Stilistik moderner Musiknotation durch mathematische Remodellierung*. Osnabrück: epOs.ISBN 978-3-940255-88-4.

Lepper, M., Trancòn y Widemann, B., & Oehler, M. (2022). *funCode 1.0 Technical Report*. Universität Osnabrück. https://doi.org/10.48693/28.

Lewin, D. (1982). A formal theory of generalized tonal functions. *J Music Theory*, *26*(1), 23–60. http://www.jstor.org/stable/843354.

Lewin, D. ([1984]2006). Amfortas prayer to titurel and the role of d in parsifal. In: *Studies in Music with Text*. Oxford University Press. ISBN 978-0-19-531713-8.

Lewin, D. ([1987]2007). *Generalized musical intervals and transformations*. New Haven: Yale University Press. ISBN 978-0-19-531713-8.

Maler, W. (1931). *Beiträge zur durmolltonalen Harmonielehre*. München: Leuckart.

Marschner, F. (1894). *Die Klangschrift*. Wien: Selbstverlag.

Mazzola, G. (1990). *Geometrie der Töne: Elemente der mathematischen Musiktheorie*. Basel: Birkhöuser.

Mitchell, W. J. (1962). The study of chromaticism. *J Music Theory*, *6*(1), 2–31. http://www.jstor.org/stable/843257.

Nápoles López, N., & Fujinaga, I. (2020). Harmalysis: A Language for the Annotation of Roman Numerals in Symbolic Music Representations. In: De Luca E and Flanders J (eds.) *Music Encoding Conference Proceedings 2020*. Humanities Commons, pp. 83–85. doi:10.17613/380x-dd98.

Neuwirth, M., Harasim, D., Moss, F. C., & Rohrmeier, M. (2018). The Annotated Beethoven Corpus (abc): A dataset of harmonic analyses of all Beethoven string quartets. *Frontiers In Digital Humanities* https://www.frontiersin.org/articles/10.3389/fdigh.2018.00016/full.

Oettingen, A. (1913). *Das duale Harmoniesystem*. Leipzig: Siegel.

Ploeger, R. (1990). *Studien zur systematischen musiktheorie*. Lilienthal, Bremen: Eres. ISBN 3872040952.

Rahn, J. (1980). *Basic atonal theory*. New York: Longman.

965)]rameau Rameau, J. P. ([1722]1965). *Traitè de l'harmonie*. New York: Broude.

Riemann, H. (1877). *Harmonische Syntaxis. Grundriß einer harmonischen Satzbildungslehre*. Leipzig: Breitkopf und Härtel.

Riemann, H. (1880). *Skizze einer neuen Methode der Harmonielehre*. Leipzig: Breitkopf und Härtel.

Riemann, H. (1882). *Die Natur der Harmonik*. Leipzig: Breitkopf und Härtel.

Riemann, H. (1918). *Handbuch der Harmonielehre*. Leipzig: Breitkopf und Härtel.

Schenker, H. (1906). *Harmonielehre*. Wien: Univeral Edition.

Schoenberg, A. (1922). *Harmonielehre*. Wien: Universal-Edition. ISBN 3-7024-0029-X.

Schoenberg, A. (1954). *Die formbildenden Tendenzen der Harmonie*. Mainz: Schott.

Tymoczko, D. (2011). *A geometry of music: Harmony and counterpoint in the extended common practice*. Oxford: Oxford University Press. ISBN ISBN 978-0-19-533667-2.

UML (2022). Unified modelling language – class diagram https://en.wikipedia.org/wiki/Class_diagram.

Vogel, M. (1975). *Die Lehre von den Tonbeziehungen*. Bonn-Bad Godesberg: Verlag für systematische Musikwissenschaft.

Weber, G. (1817). *Versuch einer geordneten Theorie der Tonsetzkunst*. Mainz: Schott.

Wikipedia contributors (2021). Context-free grammar — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Context-free_grammar&oldid=1055322802. [accessed 11 February 2022].

Wikipedia contributors (2022). Regular expression — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Regular_expression&oldid=1070604585. [accessed 11 February 2022].