FOMUS, a Music Notation Software Package for Computer Music Composers

David Psenicka
School of Music, University of Illinois
dpsenick@uiuc.edu

Abstract

FOMUS is a software tool that facilitates the process of representing algorithmically generated data as musical notation. It does this by spelling notes, quantizing offsets and durations, laying out information into voices and staves and making many other decisions such as where to include clef changes or octave transposition markings. A computer music composer using this software can work while being less concerned with the tedious difficulties involved in importing data into a graphical notation program. Information such as dynamic markings and articulations can also be generated and made to appear in the score. The program is written in Lisp, interfaces with Common Music and outputs files suitable for rendering with LilyPond or CMN or importing into a program that reads MusicXML format files such as Finale or Sibelius.

1 Introduction

Converting data generated by computer algorithms into musical notation can be a tedious procedure that is prone to error. Conventions exist in musical notation that are often not a concern for composers until it is time to render information into a notated format suitable for musicians to read and interpret. Such conventions might include barlines, note accidentals, rhythmic spelling, and other elements that are typically too difficult or complicated to bother with when the main concern is generating information more directly related to audible results.

A typical method of importing data into a popular graphical notation program such as Finale or Sibelius might involve using MIDI files to import information into that program. This procedure is rather limited, since MIDI information is constrained to pitch and duration and contains no provisions for other important notational elements such as articulation and dynamic markings. The MIDI import features for these programs are adequate at best, and might not properly import rhythms more complex than a triplet. Note spellings are usually much simpler than what is desirable. Much work may also be involved in double-checking them to insure that the score is accurate.

Most other methods of importation involve similar limitations or require the user to specify detailed elements such as accidentals, tied notes, or barlines that are too cumbersome to deal with until one is working directly with the score. FOMUS (FOFormat MUSic) offers an alternative to having to deal with these limitations, providing algorithms that make decisions regarding note spellings, voice separation, rhythmic spelling, etc.. It organizes the results into a format suitable for processing by a graphical notation program such as LilyPond (Nienhuys and Nieuwenhuizen 2003), Common Music Notation (Schottstaedt), Finale or Sibelius. Importing into Finale and Sibelius is done via the MusicXML file format (Good). Once saved or imported into one of these programs, much less work is required to adjust the results into a finished, notated score.

FOMUS is available for free at http://common-lisp.net/project/fomus/ and is licensed under the LGPL. It currently runs in SBCL, CMUCL, LispWorks and Allegro CL in the Linux, OS X and Windows operating systems. It interfaces with Common Music (Taube 1997) and is also included with the PWGL/ENP visual programming environment (Laurson and Kuuskankare 2002).

2 Related Work

Solutions to several of the problems mentioned in this paper such as automatic note spelling and voice separation have been proposed. A dissertation by Jürgen Kilian (2004) summarizes several approaches to quantization, note spelling and voice separation (or “streaming”) and describes his own system for solving these issues. The note spelling algorithm described here is similar to an algorithm developed by Emilios Cambouropoulis (2001) which bases note spelling decisions on minimal use of accidentals and avoidance of uncommon intervals such as augmented and diminished ones.

A paper by David Meredith (Meredith 2003) offers a summary of this and several note spelling techniques including his own. Approaches to voice separation are found in Cambouropoulis (2000), Chew and Wu (2004), Kilian and Hoos (2002), and in the The Melisma Music Analyzer program by Daniel Sleator and Davy Temperley (Sleator and Temperley). FOMUS differs somewhat from these algorithms in that it is meant to provide a general solution for any input data that is encountered, whether it represents music that is sparse or dense, polyphonic or homophonic, tonal or atonal, etc.. It is a practical application designed to make the composer’s task easier rather than provide accurate transcriptions of classical or popular music.
3 Features

FOMUS automates the following tasks:

- Quantizing (choosing rhythms and tuplets that minimize the amount of error between input offsets and notational offsets)
- Note spelling (semitones and quarter-tones are currently supported)
- Distribution of notes into voices and staves
- Proper note splitting and tying (rhythmic spelling), taking into account meter subdivisions, dotted rhythms, and special cases such as syncopated rhythms
- Part grouping and ordering (given the instruments for each part and an ensemble type such as an orchestra)
- Transpositions and checking of instrument ranges
- Decisions regarding placement of clef changes and octave transpositions
- Layout of articulations, slurs, beams, dynamics and text markings, and other notational elements
- Many other minor but helpful tasks (beaming, applying cautionary accidentals, etc.)
- Output to several different file formats: LilyPond, CMN, MusicXML and MIDI.

The program is designed with modularity in mind so that any of these operations will eventually be performed by choosing an algorithm from among several similar ones. Users will be able to choose the note spelling algorithm that gives the most desirable output for their own work. Many of these operations can be turned on or off if the user wishes to make certain decisions him/herself. Many automated decisions can also be overridden, in which case FOMUS uses and incorporates the user’s information when making its own decisions.

The following subsections give an overview of how FOMUS handles some of these tasks.

3.1 Input

Data is input either through Lisp expressions or a text file. Figure 1 shows a simple example using Lisp expressions. The INIT function in the first line specifies that the output file is to be a LilyPond file and that it is to be viewed immediately. Many other global options may be specified in this line. The NEWPART function specifies a part with an ID of 1 for notes to be attached to. The :piano symbol tells FOMUS that it is to notate a part for a piano instrument using a grand staff with treble and bass clefs and octave transposition signs where appropriate. Two notes are then specified—the first a dotted eighth note and the second an eighth note (the default beat duration is a quarter note, which can be changed). The marcato and staccato indications are examples of “marks,” symbols that designate a variety of articulations, ornaments, dynamic markings, and special notation instructions that can be attached to any note event. The EXEC function then sends the information to FOMUS for processing and outputs a file for importing or compiling using one of the notation programs mentioned above (in this case, LilyPond). Users can also send information to FOMUS via Common Music’s EVENTS function. The same information can also be stored in a text file which is then processed by FOMUS, an option for users who don’t want to program in Lisp or use Lisp syntax to specify events.

Figure 1: FOMUS Input Example

FOMUS decides whether to divide the 5/8 measure into 3+2 or 2+3 eighth notes, whether to spell note 66 (F#/Gb above middle C) with a sharp or flat, and which staff or staves to place the two notes on. Figure 2 shows the result of executing the contents of Figure 1.

Figure 2: LilyPond Output

3.2 Quantizing

FOMUS quantizes note offsets and durations by searching through all possible combinations of rhythms and tuplets in each measure and finding the ones that deviate the least from the user’s input values. This is done according to a quantize duration and maximum tuplet value specified by the user. Note durations that are quantized to a value of zero become grace notes. The user may also specify whether to minimize the average error or the least-mean-squared error between input and notation. One may also specify exact offsets and durations in the form of integers and ratios, in which case no quantizing is necessary and rhythms are spelled exactly as specified.

The search process described above is similar to the search used in the following subsection on rhythmic spelling and uses the same rules and constraints for determining what is or isn’t allowed. Each measure is recursively divided into smaller and smaller segments and the best solutions (the ones with minimal quantization error) are combined into larger solutions to form a complete solution.

3.3 Rhythmic Spelling

One of FOMUS’s basic operations is to decide on how to split and tie notes within a measure so that the result is properly notated according to the given time signature. Rests are added automatically. The user must specify time signature
change events which contain information such as what note duration equals a beat, how measures are to be divided (a 7/8 measure, for example, may be divided into either 3+2+2 or 2+2+3 or may always be divided into 2+3+2), and whether or not the time signature represents a compound meter. FOMUS uses this information to determine the simplest combinations of ties and tuplets that precisely represent the rhythms from the quantizing stage mentioned above. Decisions regarding metric subdivisions are made consistently across all parts, and special cases are also accounted for (for example, syncopated patterns in the form of quarter-half-quarter or eighth-quarter-quarter-eighth). Nested levels of tuplets are also possible.

The rhythmic spelling algorithm works by dividing each measure recursively into smaller and smaller segments and finding the simplest valid combination of notes, ties and tuplets in each segment. As smaller solutions are found they are combined into larger solutions to form a complete solution for the entire measure.

Once rhythmic spelling decisions have been made, FOMUS generates beaming information so that the output properly represents the decisions made regarding metric subdivisions.

### 3.4 Note Spelling

Note spellings are determined by searching through all the note events in each part and choosing the accidentals that score the highest in terms of forming easy-to-read diatonic intervals and appropriately spelled ascending or descending lines (ascending lines with sharps or descending lines with flats). Quarter-tone spellings are also possible, in which case scoring also additionally depends on whether similar inflections (quarter-tone sharps or flats) appear in close proximity. Users may also provide their own note spellings, overriding FOMUS’s decisions. Scales and key signatures are not yet included in the algorithm’s decision making, though this will be added in future versions of the program.

The search process described above allows decisions to be made over extended sequences of notes or chords. FOMUS backtracks and evaluates different possibilities until it finds the best combinations. A spelling decision takes into account all notes within a certain distance (calculated exponentially with respect to both duration and pitch) and weighs all scores according to their distances. A similar kind of search is used in the voice and staff separation algorithms described below, and can be adjusted to produce better quality output at the expense of computation speed.

FOMUS also provides algorithms for handling cautionary accidentals and other details such as how accidentals are handled when they appear beneath octave transposition signs.

### 3.5 Voice and Staff Separation

A similar type of search is used for separating voices within a part. Choices are made in the context of note events and chords occurring within a certain temporal distance. Overlapping voices are always separated if this is not allowed for the given instrument, though for some instruments (such as the piano) these are permitted and converted into chords as appropriate. Scoring is based on whether a note event is higher or lower than surrounding note events in other voices, whether a note is simultaneous with, overlapping, or adjacent to notes in the same or other voices, how smooth the “voice leading” is, how closely spaced together the chords are, and how “well balanced” the voices are (the algorithm tries to put an equal amount of material in each voice).

FOMUS also lays out voices onto multiple staves, if required, adding clef signatures and octave transposition signs when necessary. This is a separate algorithm but is again done by searching through possible solutions and using scores to find the best result. Scoring in this case is based on the number of clef changes that occur in close proximity, the number of staff changes in close proximity, the number of simultaneous voices on each staff, and the current “clef order” (a bass clef in an upper staff simultaneous with a treble clef in a lower staff receives a severe penalty, for example).

### 3.6 Other Features

FOMUS contains many other features, such as a database of common orchestral instruments and the information necessary for notating them properly (for example, the clefs used, their transpositions, pitch ranges, etc.). Users may specify one of these default instruments or define their own. Percussion instruments with little sustain and other special articulations (such as pizzicato strings) are notated with simpler durations that are easier to read. Time signature changes are automatically adjusted to avoid awkward “1/8” or other such meter changes that might occur. Multiple voices within a part are combined into chords where appropriate. Also, some articulations and ornament marks are treated specially, such as mordents or trills with accidentals above or below them (FOMUS notates them with the correct accidental when the user supplies the second pitch), pizzicato and arco markings (FOMUS sorts these out automatically) and artificial harmonics.

### 3.7 Backends

Four backends are currently supported: LilyPond, CMN, MusicXML and MIDI. The LilyPond (Nienhuys and Nieuwenhuizen 2003) typesetting program is excellent for immediately processing and displaying notation that is well formatted and readable. Common Music Notation runs natively in Lisp, and is capable of displaying all of the same notational elements. MusicXML (Good) is suitable for importing notation into commercial notation programs like Finale, Sibelius and Igor Engraver. A MIDI backend is available via Common Music and is useful because it is fairly customizable. The user can provide his/her own functions to translate notation into MIDI output, thus “personalizing” the output to a great extent to produce a better rendering than what other programs might produce on their own.

Figures 3 and 4 show how FOMUS notates a short, randomly generated piece. The Lisp expression contains a loop structure that collects random note events and passes them to
the FOMUS function for processing. The output was generated on the fly with LilyPond.

(fomus
:backend ':lilypond :view t)
:ensemble-type :orchestra
:default-beat 1/4
:global (list (make-timesig :off 0 :time '(3 4))
(make-timesig :off 7 :time '(5 8)))
:parts (list (make-part
:name "Piano"
:instr :piano
:events (loop
for basenote in '(54 42)
nconc (loop for off = 0 then (+ off dur)
and dur = (/ (1+ (random 4)) 2)
while (< (+ off dur) 12)
collect (make-note
:voice '(1 2)
:off off
:dur dur
:note (+ basenote
(random 25))
:marks (when (= (random 2) 0)
':(:accent))))))

Figure 3: FOMUS Input Example

Figure 4: LilyPond Output

4 Conclusion and Future Work

FOMUS has potential for uses other than music composition. Coupled with Common Music, it is an excellent tool for immediate viewing of musical data stored in the MIDI file format. Its modular structure allows for the addition of functionality that would be useful for music transcription. For this reason, modules for beat, meter and key detection are planned for future releases, as well as more specialized algorithms for note spelling and voice separation. Following are some planned additions and improvements:

- Combining of separately notated sections into one score
- Support for polymers
- Support for proportional notation
- Better support for microtonal pitches (depending on how these are supported in the backends)
- Algorithms for detecting meter divisions and tonal areas for automatic placement of time and key signatures
- Support for more backends including SCORE, Guido and ABC.

The intention of FOMUS is to remove the difficulty involved in rendering algorithmically-generated data into readable music notation. When relying on methods of importing notational data that only support limited features (such as the case of using MIDI files described above), a composer is forced to work within the bounds of that method. Using MIDI files as a means of transporting data, for example, encourages composers to only work with pitch and duration, and may influence the types of rhythms that are composed depending on how easily they can be imported into a certain notation program. FOMUS is designed to remove many of these limitations and make important aspects of notation easily accessible and controllable within a computing environment. The composer can then focus less on how to arrange his/her score and more on the process of making music.

References


