**Cordelia Hall and John T. O'Donnell**
School of Computing Science
University of Glasgow
Glasgow, G12 8QQ, UK
{cordelia.hall, john.odonnell}
@glasgow.ac.uk

# Regular Expressions as Violin Bowing Patterns

**Abstract:** String players spend a significant amount of practice time creating and learning bowings. These may be indicated in the music using up-bow and down-bow symbols, but those traditional notations do not capture the complex bowing patterns that are latent within the music. Regular expressions, a mathematical notation for a simple class of formal languages, can describe precisely the bowing patterns that commonly arise in string music. A software tool based on regular expressions enables performers to search for passages that can be handled with similar bowings, and to edit them consistently. A computer-based music editor incorporating bowing patterns has been implemented, using Lilypond to typeset the music. Our approach has been evaluated by using the editor to study ten movements from six violin sonatas by W. A. Mozart. Our experience shows that the editor is successful at finding passages and inserting bowings; that relatively complex patterns occur a number of times; and that the bowings can be inserted automatically and consistently.

A significant challenge faced by a string player learning a new piece is deciding on the bowings to be used. String players spend a significant amount of practice time creating and learning bowings. For this reason, teachers generally provide bowings for students until they are experienced enough to create their own. These may be indicated in the music using up-bow and down-bow symbols, but those traditional notations do not capture the complex bowing patterns that are latent within the music. Orchestral players also create bowings, agreed upon by the concertmaster and conductor, yet these may vanish when a rented orchestral part is cleaned up. String musicians know a great deal about bowings, but much of their knowledge is ephemeral and not permanently integrated into our musical heritage.

Many music-editing software tools offer little help with bowings. They accept individual slurs, ties, and up-bow and down-bow symbols sprinkled throughout the text, but do not treat a complete bowing pattern as an object in its own right. This limits what the musician can do. For example, it is impossible to search for a specific bowing, or to look for passages where a similar approach would be suitable.

In this article, we introduce explicit notation for bowing patterns, as well as a software tool that uses the patterns to give practical support to string players. Patterns can be used to define a class of passages that can be bowed in a particular way, and they can also specify the bowings to be applied in matching passages. Bowing patterns help a string player to bow similar passages consistently. Patterns allow bowings to be treated as entities so that they can be defined, searched for, applied to music, stored in a database, and analyzed.

Patterns are defined using regular expressions, a mathematical method for precisely defining the structure of a class of strings. This mathematical precision leads to clarity for the musician, and it provides a foundation for algorithms that operate on bowings. We write these patterns in both regular expression notation and using traditional music score notation. The score notation could be used directly by musicians without requiring that they learn how to write regular expressions, e. g., when editing orchestral parts on an electronic music stand. The patterns can be used both to search for suitable passages of music and to insert a bowing into the matching positions. This enables software to edit parts for different string sections in an ensemble so that their bowings are compatible, as well as to archive bowings in a database.

To demonstrate the feasibility of this approach, we have implemented a prototype software music editor that implements bowing patterns. The music is entered in Lilypond notation (Lilypond is free software for typesetting music), and the regular expression bowing patterns are entered as character strings. The system can search the music for occurrences of a pattern, and it can insert bowings consistently. The software annotates the Lilypond notation accordingly, and uses Lilypond

Computer Music Journal, 36:2, pp. 74–84, Summer 2012

to typeset the music with the bowing symbols inserted.

Our approach has been evaluated by using the editor to study ten movements from six violin sonatas by W. A. Mozart. Our experience shows that the editor is successful at finding passages and inserting the bowings; that relatively complex patterns occur a number of times; and that the bowings can be inserted automatically and consistently.

## Related Work

Several computer systems have used regular expressions to search music. Humdrum (Huron 1994) is a set of software tools for processing music, and it supports accessing representations of music with regular expressions. The full representation of a piece of music contains many different kinds of information (e.g., fingerings) that are not relevant to bowing. Humdrum makes it possible to perform a preliminary filtering to remove any irrelevant annotations in the music representation. This would be helpful for using bowing patterns, as the regular expressions would not need to account for irrelevant annotations. Humdrum uses a set of standard, general-purpose Unix tools, however, and it relies on the standard syntax for regular expressions. This generality makes the notation more complex than needed just to process bowings, and would be a barrier for musicians who are not also computer scientists.

Regular expression–style searching has been used in music information retrieval for polyphonic music (Dovey 2001). The music is represented as a matrix similar to a piano roll, and the search algorithm matches portions of the music using regular expressions. In Dovey's system, the regular expressions are part of the mechanism that implements the search, and they are not manipulated directly by the user. This differs from the approach presented in this article, where the string player specifies bowings explicitly as regular expressions.

Bowing patterns could be integrated into music editors that are sufficiently extensible. Lime is a general music editor that contains a music representation language called Tilia (Haken and Blostein 1993). The user does not directly work with Tilia; the language enables Lime to support "smart," customized, high-level music editing operations. These have a wide range of applicability, such as generating specialized document types (e.g., a score and its parts) or generating a piano reduction. Tilia is extensible, and supports the use of temporary nodes annotated with information that may be newly defined, and which are created and destroyed during an editing session. The Lime formatter uses these to keep track of information generated during early passes that is used by later passes. The bowing patterns discussed in this article could be incorporated into Lime using Tilia; thus, it would be possible to perform a preliminary analysis of bowings and use the results to annotate the music.

The flexibility of the Lime editor also supports the application of graph grammars to recognize complex structures in music (Fahmy and Blostein 1993). A low-level graph, which represents all details of the music, is transformed into a high-level graph that abstracts away information that is irrelevant to a particular analysis. This approach could be adapted to support analysis of bowings using regular expressions.

PWGL (Laurson and Kuuskankare 2004) is a visual programming language that supports composition and sound synthesis, and has been used to develop specialized music editors. These editors are also visual, and use a graphical user interface rather than text. This approach to music editors could support the design of a graphical editor for bowing patterns.

A variety of multimedia solutions for aspects of music, including performance, gesture analysis, score following, and others, are integrated in the i-Maestro system (Ng and Nesi 2008). The system can observe a performance by a musician in real time and automatically annotate the score with bowing symbols. This is achieved through a combination of techniques, including analysis of the score and motion capture. It would be useful to incorporate bowing patterns in such a system.

An electronic music stand allows software to adapt music notation as it is displayed on the screen. One such system is MOODS (Bellini, Fioravanti, and Nesi 1999), which manages music

for performers and orchestras. MOODS provides real-time support for a variety of requirements, such as the conductor, string section leaders (who may work together to decide what bowings are used), string section players, and a music archivist. The system uses rules to arrange musical symbols on screen automatically, so that the display suits the requirements of the musician. This would be an ideal environment for helping a musician to work out a bowing using bowing patterns.

Bowing annotations have also been supported in digital music stands such as muse (Graefe et al. 1996) and eStand (Cross 2004). Searching for bowing annotations as bowing patterns does not appear to be supported. A key motivation for developing digital music stands is their potential to process the annotations produced by musicians as they work. These annotations are valuable because they reflect high-level musical expertise that is commonly destroyed when rented music is returned (Bellini, Fioravanti, and Nesi 1999; Winget 2006).

Music can be analyzed and stored in a database (Rigaux and Faget 2011), applying computer science database technology to the problems of music information retrieval. Database systems normally use the Structured Query Language (SQL); as SQL supports regular expressions (Eisenberg and Melton 1999), bowing patterns could be used in music information retrieval.

## Regular Expressions and Bowing Patterns

The software system uses bowing patterns, described in this section, to search the music for passages where a particular bowing could be used, and (optionally) to insert that bowing. Before the search occurs, the system needs to know the original bowing—that is, which notes are played on each bow stroke. The music notation that is provided as input must give the duration and pitch for each note, and may contain specific up-bow or down-bow annotations for some notes. The software begins by analyzing the music and calculating the complete bowing, so that each note is explicitly marked as being up-bow or down-bow, and each bow stroke is applied to a specified note or sequence of notes. This is done by an algorithm that follows common conventions: (1) the bow changes directions on each note or group of notes; (2) a sequence of notes that are slurred are played on one bow stroke; (3) after a rest, a note at the beginning of a bar is played with a down-bow; and (4) a pickup note is played with an up-bow. Any explicit up-bow or down-bow annotations are followed, and override the default predictions. The result of this algorithm is a fully annotated version of the music, which is kept in a data structure in the computer's memory. It is possible to print out the music with all of the calculated annotations, or some subset of them.

We want to write two kinds of bowing patterns: (1) a description of the pattern of up- and down-bows used in a passage of music, and (2) a description of how to take an existing bowing and modify it systematically. Before introducing the formalism for specifying these patterns, we first consider the requirements that the notation must satisfy.

We need to find a notation for describing an entire bowing pattern as an entity to itself, not just as a set of symbols scattered through the text. Consider the requirements of such a notation. It must be able to (1) specify specific types of notes with their bowings (e.g., a quarter note on an up-bow); (2) concatenate several smaller patterns to form a larger one (e.g., a quarter note on an up-bow, followed by a quarter note on a down-bow); (3) provide the ability to search the music for either of the two alternative bowings (e.g., a choice between a quarter note on a down-bow, or two eighth notes both on a down-bow); and (4) support a repetition of a pattern (e.g., a pattern for bowing a sequence of sixteenth notes consisting of a down-bow followed by an up-bow that is to be repeated over a long passage).

Thus, the musical requirements suggest that a bowing pattern consists of a set of basic symbols, along with three ways to combine smaller patterns to form a larger one: concatenation, alternation, and repetition.

In computer science, the type of formal language called a regular expression has exactly the characteristics needed. Regular expressions have been studied extensively, are used in many applications, and are supported by many software tools. Therefore, we use regular expressions to define bowing patterns. We

will define regular expressions and then explain, in detail, how they are used to define bowing patterns.

We can define a bowing pattern precisely by means of a formal language specified using a regular expression. A formal language defines a set of *sentences* built from an *alphabet*, according to a precise set of rules called a *grammar*. The grammar defines precisely which sentences are in the language, as well as giving the structure of each sentence. A *regular expression* is a simple form of grammar that defines sentences with a nested structure; this structure is suitable for describing bowings.

We now introduce some necessary notation. Let the alphabet $A$ be a fixed set of symbols (these will include, for example, the up-bow symbol and many others). A *string* is a sequence of symbols of the form $x_1 \quad x_2 \ldots x_k$ where each symbol $x_i$ is in the alphabet $A$. A *sentence* is a string that obeys a set of grammatical rules that will be given subsequently. Thus, a string might be a nonsensical sequence, or it could be a grammatically correct sentence. Each bowing pattern will be represented by a grammatically correct sentence. We specify the set of grammatical sentences by means of a set of rules; each rule is written in a metalanguage, and it specifies a set of sentences. The rules are as follows.

**Rule 1: Literal**

For any symbol $a$ in $A$, the regular expression $a$ denotes the set of strings $\{a\}$. (The symbols used for bowing patterns are defined subsequently.) For example, $d$ is the symbol for a down-bow, so $d$ is also a regular expression that defines a set consisting of one sentence (which is $d$). The literal rule allows us to put basic symbols into a pattern.

**Rule 2: Concatenation**

Suppose that $X$ is a regular expression, and also suppose that $Y$ is a regular expression. Then $XY$ is a regular expression. (It is often called "the concatenation of $X$ and $Y$.") The expression $XY$ denotes $\{x_1 x_2 \ldots x_i y_1 y_2 \ldots y_j\}$ such that $\{x_1 x_2 \ldots x_i\}$ is a member of the set denoted by $X$ and $\{y_1 y_2 \ldots y_j\}$ is a member of the set denoted by $Y$. In other words, $XY$ describes sentences that consist of any sentence from $X$ followed by any sentence from $Y$. For example, 8$u$ is a concatenation of the regular expression 8 with the regular expression $u$, and it denotes the set of sentences {8$u$}. Naturally, if $X$ and $Y$ are expressions denoting many sentences, then their concatenation will denote even more possible sentences. The concatenation rule allows us to follow one bowing pattern by another.

**Rule 3: Alternation**

Suppose that $X$ and $Y$ are regular expressions. Then $X \mid Y$ is a regular expression called "the alternation of $X$ and $Y$." It denotes a choice between $X$ and $Y$. The set of sentences denoted by $X|Y$ is the union of the set of sentences denoted by $X$ and the set denoted by $Y$. For example, 8$u$|16$u$ denotes the set {8$u$, 16$u$}. Alternation allows us to say that there are several possible bowing patterns, and one of them must be chosen (e.g., in a passage of music with alternate endings, we would use alternation to choose between the bowing pattern for the first ending and the bowing pattern for the second ending). For syntactic reasons, when we write bowing patterns we will put braces around the two alternatives. Therefore, if $p$ and $q$ are bowing patterns, then their alternation would be written $\{p \mid q\}$. Note that the symbols "|", "{", and "}" are part of the meta notation for defining regular expressions, and they are not symbols in the alphabet. (In other words, these symbols are not musical annotations themselves.)

**Rule 4: Repetition**

Suppose that $X$ is a regular expression. Then $X^*$ is a regular expression called "the repetition of $X$." The expression $X^*$ denotes a set of sentences consisting of zero or more sentences chosen from the set denoted by $X$. (Informally, the repetition of $X$ is like an arbitrary number of concatenations of $X$.) For example, $\{8d|16u\}^*$ denotes an infinite set of sentences, including 8$d$, 8$d$ 16$u$, 16$u$ 8$d$

16*u*, and more. For syntactic reasons, we will use square brackets around the regular expression being repeated, so the repetition of $p$ is written $[p]$. Repetition allows us to define a bowing over a long passage of music by concatenating as many patterns for small parts of the passage as needed. Naturally, repetition is useful when the long passage contains many repetitions of similar bowings. Note that "[" and "]" are part of the meta notation; they are not part of the alphabet of musical annotations.

**Bowing Patterns**

We now define bowing patterns using regular expressions. A bowing pattern must express the essential aspects of a passage while omitting irrelevant aspects. Therefore, we will describe a note with only its duration and the direction of the bow to be used when it is played.

A bowing pattern will apply to a sequence of notes, so the pattern must be able to express concatenation. Sometimes bowing patterns start or end in slightly different ways in various places; this means that the notation must be able to express a choice among alternatives. A regular bowing may occur several times within a passage, and it is even possible for the number of repetitions to vary in several places within the music. Therefore, the notation must be able to express the repetition of a smaller pattern. For these reasons, regular expressions are suitable for expressing bowing patterns.

The alphabet is the union of the set $C$ of basic note durations, the set {.} for dotting a duration, the set $B$ of bow directions, and the set of articulation symbols.

$C = \{1,2,4,8,16,32\}$ is a set of symbols that specify the duration of a note (i.e., whole note through thirty-second note). A dot can be added as needed to increase the duration by 50 percent, in accordance with traditional music notation. A duration $D$ is a regular expression that specifies a basic note value (taken from $C$), optionally followed by a dot, which has the traditional musical function of augmenting the duration by a half. Thus $D = C \mid C.$. The set $B = \{u, d\}$ is the set of bow directions, corresponding to up-bow and down-bow, respectively. A stroke is specified by the regular expression $S = DB$; that is, a duration followed by a direction. For example, $S$ contains the note representations {1*u*, 1*d*, 1.*u*, 1.*d*, . . . ,2*u*, 2*d*, 2.*u*, 2.*d*, . . . }.

A common symbol in standard music notation is a curved line over a sequence of notes. This symbol serves several distinct functions. It may be used to combine two adjacent notes into a single note with a longer duration; when used this way, the curve is called a *tie*. The curve is often used in string music to indicate that a group of notes are to be played legato on a single bow stroke; we call this usage an *articulation slur*. A curve may also be placed over a group of notes in order to indicate phrasing, without necessarily implying that the notes have to be played on one bow stroke. We call this usage a *phrasing slur*. We need to make these distinctions, as the software needs to know exactly which notes are played on each bow stroke. Therefore, we introduce a set of paired symbols that can be placed around a sequence of notes: the angle brackets "<" and ">" denote an articulation slur and the round brackets "(" and ")" indicate a phrasing slur. Additionally, the character "`" denotes a staccato mark, and "-" denotes a tenuto mark. Because angle brackets and round brackets are used for slurs, we will use square brackets, "[" and "]", to denote groupings of regular expressions.

Finally, there are the symbols "!", "$", and "%", each of which is used for editing bowing patterns. These symbols represent information about a symbol or note representation. When a symbol, such as "(", is prefixed by "$", forming $(, it means that the symbol should be added to the pattern. If "!" is applied to a note representation, such as !2*u*, it means that the note representation was originally 2*d* and has been changed to 2*u*. Thus, "!" changes the direction of a bow stroke. If "%" just appears by itself, then it means that the symbol to which it has been matched by the editor is deleted.

A bowing pattern is a regular expression over the alphabet. For example, the concatenation 16*u* 8*d* represents a sixteenth-note up-bow followed by an eighth-note down-bow; the alternation {16*u*|8*d*} represents a sixteenth-note up-bow or an eighth-note down-bow, and the repetition [8*d*] represents a sequence of eighth-note down-bows.

The pattern <!2*u* 2*u*> will match <2*d* 2*u*>, but not <2*u* 2*u*>. A more complex pattern, $(!4*u* 2*u* $), will match 4*d* 2*u* and rewrite it as (4*u* 2*u*). The effect is to turn a down-bow followed by an up-bow into one up-bow with the two notes slurred.

We have implemented a software application that processes a piece of music using these notations. The program reads input in Lilypond notation. The input may contain the extra notations described earlier, as well as standard Lilypond notation. The program has the ability to work through the entire piece of music in order to determine the bow direction for every note, so that the user only needs to put in explicit up- or down-bow symbols where needed (for example, if two articulated notes are played on separate bow strokes in the same direction). The program can search the music for sequences of notes that follow a bowing pattern (expressed by the user using the regular expression notation), and it can also insert the symbols required to apply a pattern to a passage (this may result in additional up- or down-bow symbols, or articulation slurs). The program can write out the edited music to be typeset by Lilypond, with the modified bowings indicated by the automatically inserted symbols.

The program is implemented in Haskell, a widely used pure functional programming language. It is free software, and is available at www.dcs.gla.ac.uk/~jtod/music/bowing.

The program incorporates several features that are not discussed in detail here, including a complete calculation of the bowing (so that the bow direction for every note is known). Future work includes providing a mechanism to allow the musician to specify bowing patterns using traditional music notation, but the current version of the program requires textual specification of the regular expressions.

## Analyzing Bowings in Mozart Sonatas

The system was evaluated by using it to apply 13 bowing patterns to ten movements of Mozart's piano and violin sonatas. Our approach was as follows. We studied the edition by Schradieck (Mozart 1934), identified some of his bowings, and expressed them as bowing patterns. Our editor software then searched the Urtext of the music, located the places where Schradieck's approach could be used, and changed the annotations to follow his bowing. The patterns used are as shown in Tables 1, 2, and 3; Figures 1 and 2 show the effect of editing the music according to the patterns.

**Table 1. Five Bowing Patterns Inserted by the Editor**

| *Edit* | *Location* |
|---|---|
| {2*d* \| ( 2.*d* 4*d*)}<br>$< (16*u* 8.*u* )<br>(16*d* 8.*d* )<br>{(16*u* 8.*u* ) $> \| $>} | KV 296, movt. 3,<br>7 occurrences |
| {2*d* \| ( 4.*d* 8*d*)}<br>$(8*u* !8*u* 8*u* !8*u* $) | KV 296, movt. 3,<br>5 occurrences |
| 8*d* $' $( 16*u* !16*u*<br>8*u* $) $( 16*d* !16*d*<br>{4*d* $) \| 8*d* $) } | KV 301, movt. 1,<br>6 occurrences |
| $(4.*d* !16*d* 16*d* $)<br>$(8*u* $' !8*u* $' $)<br>$(!16*d* 16*d* !16*d* 16*d* $)<br>$(4.*u* !16*u* 16*u* $)<br>8*d* 8*u* | KV 301, movt. 1,<br>2 occurrences |
| [8*d* $( 8*u* $' !8*u* $' $)<br>(!8*d* !8*d* )<br>{$( !16*u* 16*u* $)<br>\| !8*u*} ] | KV 301, movt. 2,<br>2 occurrences |

We translated ten movements from six Mozart piano and violin sonatas into Lilypond, and looked at an edition bowed by Schradieck for information on bowings to be changed and inserted. The patterns used appear in the Edit column. The Location column gives the Köchel number of the piece, the movement number, and the number of times the pattern occurs in the movement. These patterns are found in KV296 and KV301.

A significant result is that some of the patterns enabled useful edits (for inserting a bowing into the music) in several places, commonly more than once, and as many as seven times. The editor clearly saves time in editing more than one passage, some of which have repeating sub-patterns within them. Furthermore, it improves consistency: similar passages will be bowed the same way.

As examples, we look at the first and second bowing inserted on the first line of Figure 1. The

**Table 2. Five Bowing Patterns Found in KV 302, KV303, and KV304**

| *Edit* | *Location* |
|---|---|
| 16*u* 16*d* 16*u* 16*d*<br>16*u* 16*d* 16*u*<br>( 16*d* 16*d*)<br>$( 16*u* $' !16*u* $' $)<br>( !16*d* !16*d* )<br>$ ( !16*u* $' 16*u* $' $) | KV 302, movt. 2,<br>5 occurrences |
| 4*d* 8*u* $- 8*d* $-<br>( 8*u* 8*u* 8*u* 8*u* ) | KV 302, movt. 2,<br>4 occurrences |
| { ( 4.*d* 16*d* 16*d* )<br>$( 8*u* $' !8*u* $' $)<br>\| ( !4.*d* !16*d* !16*d* )<br>$( !8*u* $' 8*u* $' $) } | KV 303, movt. 2,<br>5 occurrences |
| 8*d* 8*u* 2*d* 2*u*<br>2.*d* 8*u* 8*d*<br>[ $< $( 2*u* !4*u* $) 4*u* $'<br>$ > 2.*d* 8*u* 8*d* ]<br>4*u* 4*d*<br>$( 4*u* $' !4*u* $' $) | KV 304, movt. 1,<br>4 occurrences |
| ( 8*u* 8*u* 8*u* 8*u* ) 8*d* $' 8*u* $'<br>[ ( 8*d* 8*d* 8*d* 8*d* )$( 8*u* $' !8*u* $' $)<br>( !8*d* !8*d* !8*d* !8*d* )$( !8*u* $' 8*u* $' $) ] | KV 304, movt. 2,<br>1 occurrence |

**Table 3. Three Bowing Patterns Found in KV305 and KV306**

| *Edit* | *Location* |
|---|---|
| 8*d* ' 8*u* ' 8*d* ' 8*u* ' 8*d* ' 8*u* ' 8*d* '<br>$( 8*u* % !8*u* % 8*u* % !8*u* % 8*u* % $)<br>$( 8*d* % !8*d* % $) !8*u* '<br>$( !8*d* % 8*d* % $) 8*u* ' | KV 305, movt. 1,<br>3 occurrences |
| 4*d* $( 16*u* !16*u* 16*u* !16*u* $)<br>!4*d* $( !16*u* 16*u* !16*u* 16*u* $)<br>4*d* $( 16*u* !16*u* 16*u* !16*u* $) | KV 306, movt. 1,<br>7 occurrences |
| ( 8.*d* 32*d* 32*d* $)<br>$( !8.*u* !32*u* !32*u* $)<br>$( 8.*d* 32*d* 32*d* ) | KV 306, movt. 2,<br>2 occurrences |

first bowing has a phrasing slur inserted in the second bar, which covers all of the notes in the bar (see line 2 of Figure 1). The bowing pattern used to insert this is {2*d*| ( 2.*d* 4*d*)} $< (16*u* 8.*u*) (16*d* 8.*d*) {(16*u* 8.*u*)$> | $> }. The only changes introduced are the phrasing slurs $< and $>.

The second bowing inserted on that line is done using the bowing pattern {2*d*| (4.*d* 8*d*) }$(8*u* !8*u* 8*u* !8*u*$). It starts with an alternation, because a similar passage, which we also want to edit, starts differently. The eighth notes afterwards, which match B, A, G, F, are surrounded by $(and $), indicating that new parentheses will be inserted there. Notice that the second and fourth notes have been annotated with "!", indicating that they were originally down-bows.

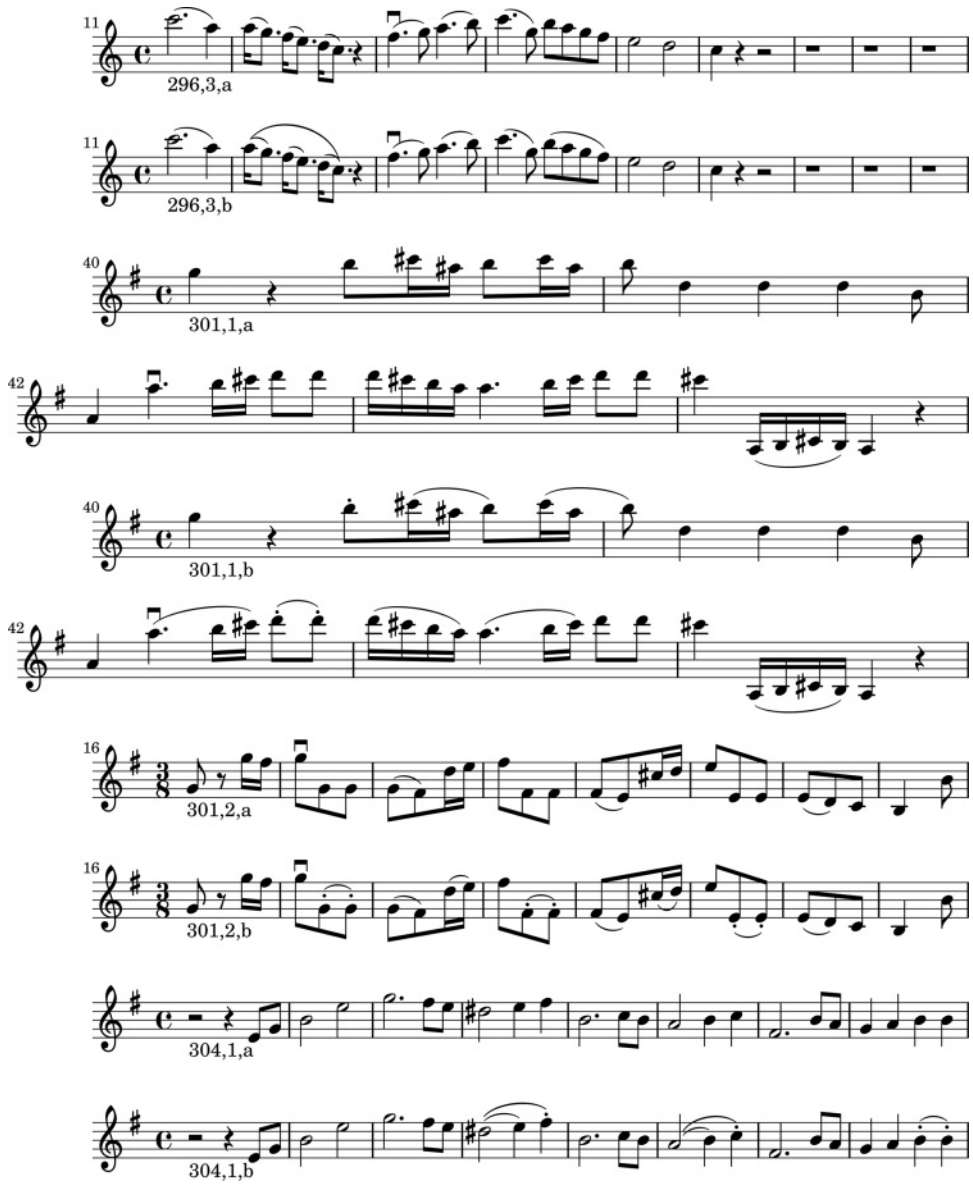
Most of the bowing patterns in these tables contain concatenations and alternations. However, there are a few containing repetitions, which are especially interesting within the context of music composition. These patterns sometimes suggest a "while" loop, in which the concatenation within the repeat is unrolled several times, followed by an alternative from an alternation, which suggests the exit of a while loop. This illustrates a way in which thematic material is often developed in music.

For example, in KV 301, movement 2, the bowing pattern has a repetition, within which there is a concatenation followed by an alternation (see Table 1 and Figure 1). When the concatenation is matched for the last time (mm. 21–22 of lines 7 and 8 in Figure 1), the other alternative of the alternation is matched, and the match of the bowing pattern succeeds. Similarly, in KV 304, movement 1 (Table 2 and Figure 1), the matching pattern is a concatenation, followed by a repetition, followed by a concatenation. The repetition captures the repeated figure in the theme of the movement.

Some of the bowing patterns occurred numerous times; the numbers of occurrences appear in Tables 1, 2, and 3.

We expect these repetitive patterns to be found often in Western string music in general, as tonal Western composition styles are derived to some extent from the period in which Mozart wrote his music. One good indication of the likely requirements of the string literature is the body of etudes and technical exercises that has grown up around it, and a (cursory) inspection of it suggests that these useful pieces also contain repetitive patterns.

*Figure 1. Part of the Lilypond output generated by the editor for the pieces referred to in Tables 1 and 2. The versions labeled "a" are the original Mozart; the versions labelled "b" are generated by the editor using the bowings of Schradieck (Mozart 1934).*




## Writing Bowing Patterns in Score Notation

Regular expressions are concise, but the notation is unfamiliar to most musicians. Therefore, we also provide a method for using traditional music notation to express bowing patterns. This requires some special notational conventions which are shown in Figure 3. Figure 4 shows the score notation corresponding to some of the regular expressions given earlier.

Each of the patterns given in Tables 1, 2, and 3 is used by the software, both to find and to edit bowings. For this reason, some practice is required to understand them, and musicians interested in using them may not have the time or patience to do so. To simplify the process of understanding, we

*Figure 2. A final example showing the effects of deleting staccato marks and inserting slurs instead.*

*Figure 3. Two bowing patterns (a) and (b), written using score notation, followed by operations over bowing patterns that use (a) and (b) to build up larger patterns.*

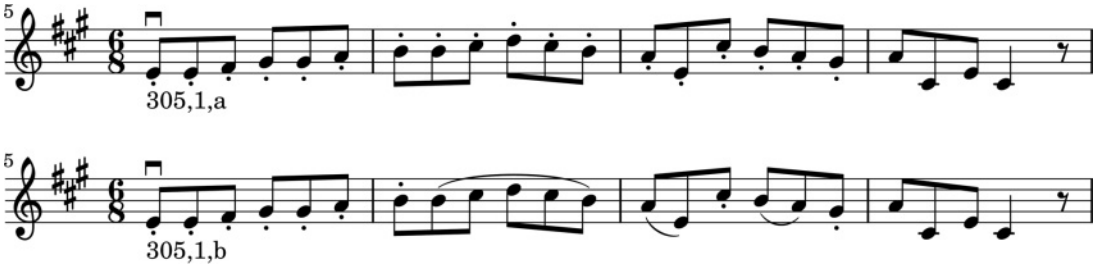



*Figure 2*

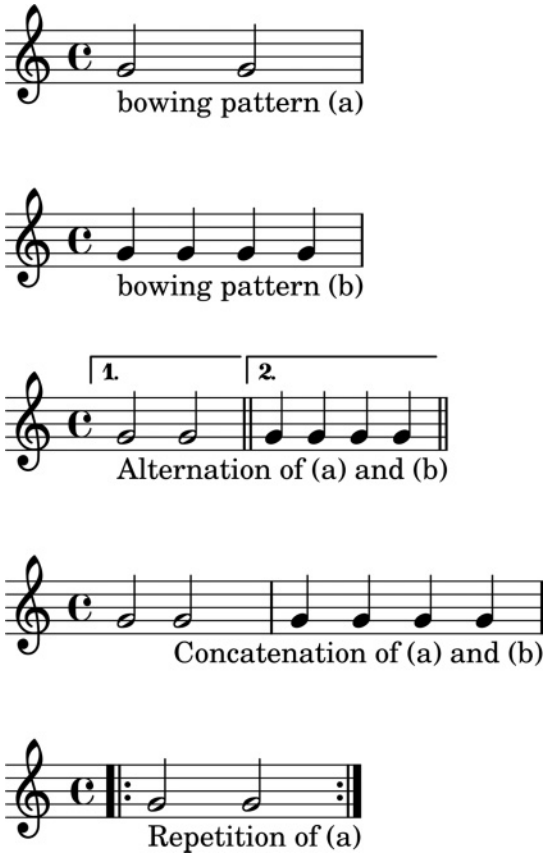



*Figure 3*

decided to split the patterns into two, the one found in the music originally being the "old" pattern, and the one used to edit that music being the "new" pattern. As bowing patterns are found in different places in the music, each note in the pattern must represent one of several possible pitches, so we decided to represent all pitches with the same note, a G above middle C. A slur denotes a bow-stroke and says nothing about the pitches of the notes being slurred, so a string player should not read these slurs as ties. A bowing symbol for each stroke is given explicitly, so that the bowing is easy to understand without having to calculate the direction of the bow.

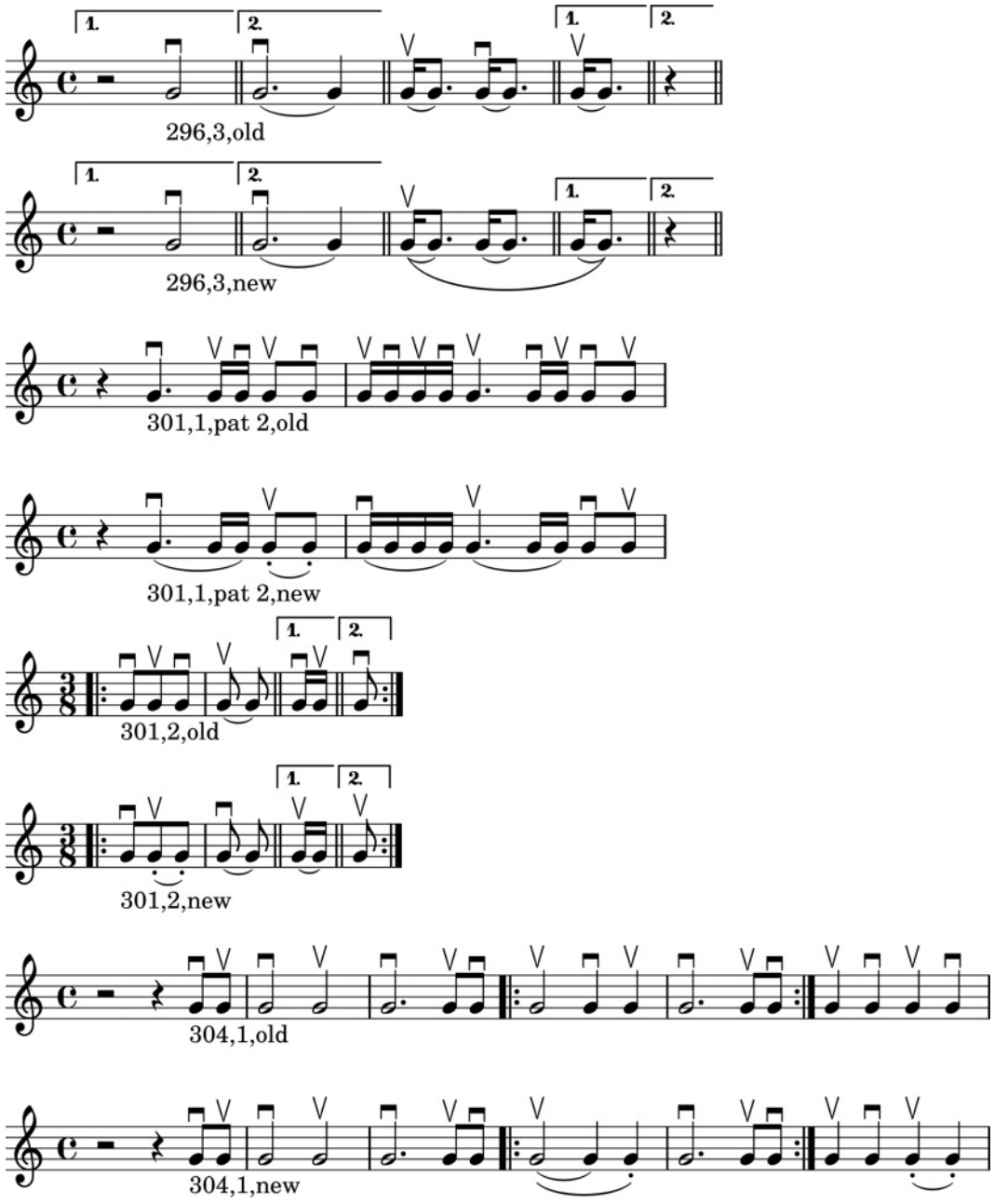
Alternations are represented by first and second endings, a concatenation by a staff containing one or more bars or bar fragments, and a repetition by the traditional repeat sign. These are shown in Figure 3, and the patterns displayed in Figure 4 use them.

Consider each set of patterns in Figure 4 in turn; Figure 1 gives passages of the music being matched and edited.

In the old pattern for KV 296, movement 3, the pattern starts with two alternate possibilities, represented as an alternation in the bowing pattern given in Table 1. The bowing could start with a down-bow on a half note, or alternatively on a dotted half note slurred with a quarter note. Whichever of these is selected, it is followed by a single bar containing the concatenation of two strokes. The first is an up-bow on a sixteenth note slurred with a dotted eighth note, the second a down-bow on a sixteenth note slurred with a dotted eighth note. Finally, the bowing pattern finishes with an alternation that has two alternatives. The first is an up-bow on a sixteenth note slurred with a dotted eighth note and the second is just a rest. The new pattern creates one large slur from the smaller slurs.

In the second pattern for KV301, movement 1, the old pattern is a concatenation of two bars. The

*Figure 4. Four bowing patterns from Figures 1 and 2 converted to score notation. We have separated each pattern into two simpler ones, labeling the first of these "old" and the second "new." The "old" pattern is found in the music by the editor. The "new" one is used to edit the music.*




new pattern introduces some slurs (see Table 1, and mm. 42–43 in Figure 1, line 6).

In the old pattern for KV301, movement 2, a repeated section one-and-a-half bars long is followed by two alternatives. The new pattern adds some slurs (see Table 1, and mm 17–18, 19–20, and 21–22 in Table 1, line 8).

In the old pattern for KV304, movement 1, there is a concatenation of two and a half bars, followed by a repeated section of two bars, and ending in

a smaller concatenation. Again, the new pattern introduces some slurs (see Table 1 and the final line of Figure 1).

## Conclusion

We have implemented a music editor that uses regular expressions to describe bowing patterns. We plan, in a future development of the software, to allow the regular expressions to be represented in traditional music notation using special conventions, which will make the system easier to use by musicians. The system allows a musician to search for similar passages and apply a bowing consistently throughout a piece of music. It has been tested by using the system to edit ten movements from six piano and violin sonatas by Mozart, and it is successful in finding passages that match the regular expressions and inserting the new bowings.

Treating a complete bowing as an entity, rather than just a number of separate annotations in the music, allows software tools to help the musician to achieve consistency. Students may find it useful to apply such software in the comparison of bowings in various editions by well known musicians.

## Acknowledgments

We would like to thank the anonymous referees, who made many useful suggestions.

### References

Bellini, P. , F. Fioravanti, and P. Nesi. 1999. "Managing Music in Orchestras." *Computer* 32(9):26–34.

Cross, J. 2004. "eStand(TM) Electronic Music Stand." *Notes* 60(3):754–756.

Dovey, M. 2001. "A Technique for Regular Expression Style Searching in Polyphonic Music." In *Proceedings of the International Society for Music Information Retrieval*, pp. 187–193.

Eisenberg, A., and J. Melton. 1999. "SQL:1999, Formerly Known as SQL3." *ACM Special Interest Group on Management of Data (SIGMOD) Record* 28(1):131–138.

Fahmy, H., and D. Blostein. 1993. "A Graph Grammar Programming Style for Recognition of Music Notation." *Machine Vision and Applications* 6(2–3): 83–99.

Graefe, C., et al. 1996. "Muse: A Digital Music Stand for Symphony Musicians." *ACM Interactions* 3(3):26–35.

Haken, L., and D. Blostein. 1993. "The Tilia Music Representation: Extensibility, Abstraction, and Notation Contexts for the Lime Music Editor." *Computer Music Journal* 17(3):43–58.

Huron, D. 1994. *The Humdrum Toolkit Reference Manual*. Stanford, California: Center for Computer Assisted Research in the Humanities at Stanford University. Available online at humdrum.ccarh.org.

Laurson, M., and M. Kuuskankare. 2004. "PWGL Editors: 2D Editor as a Case Study." In *Proceedings of Sound and Music Computing Conference* (pages unnumbered).

Mozart, W. A. 1934. *Eighteen Sonatas For Piano and Violin*. H. Schradieck, ed. New York: G. Schirmer.

Ng, K., and P. Nesi. 2008. " i-Maestro: Technology-Enhanced Learning and Teaching for Music." In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pp. 225–228.

Rigaux, P., and Z. Faget. 2011. "A Database Approach to Symbolic Music Content Management." In S. Ystad et al., eds. *International Symposium on Computer Music Modeling and Retrieval (CMMR) 2010*. Lecture Notes in Computer Science, vol. 6684. Berlin: Springer-Verlag, pp. 303–320.

Winget, M. 2006. "Heroic Frogs Save the Bow: Performing Musician's Annotation and Interaction Behavior with Written Music." In *Proceedings of the International Society for Music Information Retrieval*, pp. 73–78.