Edgar H. Sibley
Panel Editor

A music-description language designed to facilitate both electronic communication and publication-quality printing of musical scores incorporates a syntax for expressing concurrency and two-dimensionality and places new demands on text formatters.

A LANGUAGE FOR MUSIC PRINTING

JOHN S. GOURLAY

The process of transforming an original handwritten musical score into the high-quality form required for performance and publication is a time-consuming and error-prone one. Still commonly a manual process using engraving or lithography, music printing has benefited little from recent technical advances that have affected the rest of the printing industry. Manual transcription of the original manuscript is not only tedious and time consuming, but it also introduces errors that must then be laboriously removed by proofreading and manual correction of the master. For orchestral compositions, the problem is magnified since the original must be transcribed twice, once for the conductor's score and again as separate parts for individual instruments.

At the same time, the typography for other types of documents is becoming increasingly automatic. It is now typical for newspaper journalists to forgo the traditional typewriter and enter their stories directly into a computer via a display terminal. Editors then add headlines and electronically simulate page layouts without the delays and errors inherent in typesetting, cutting, pasting galley proofs, etc. Music printing, of course, is a vastly more complicated process, and it is this complexity that has prevented its automation in a practical way. Recent innovations, however, in applying computers to high-quality typography for books and technical journals make the problem of music appear much more tractable. It is now possible (but not common) for an author of a mathematical paper to submit the text to a publisher on a magnetic tape or disk. The publisher can then make any necessary editorial changes through a computer terminal and print the paper without incurring the costs of manual typesetting and subsequent proofreading and correction.

The important thing is that this difficult problem of properly formatting very complex mathematical formulas can be handled almost totally automatically at a level of quality that rivals the best manual mathematical typography. Music may be even more difficult than mathematics, but the analogies between the two are strong. Both involve the accurate placement of very large and very small characters in configurations where vertical alignment and spacing are fully as important as horizontal. Done manually, both require the attention of exceptionally qualified compositors. In the light of the recent developments in mathematical typesetting, it is clearly time to reconsider the automation of music printing.

This work was partially supported by the National Science Foundation under grant number IST-8514308.

Related Work

The large problem of computerized music printing can be looked at in terms of three rather separable components—*input*, *communication*, and *output*. The input problem involves providing a mechanism by which a composer realizes and makes permanent his or her creation. Traditionally, this is done with pencil and paper, taking the form of a quick rough draft in traditional music notation. The communication problem involves conveying the composition to collaborators, editors, or printers, again traditionally served by the handwritten draft. The output problem involves converting the composition from the form in which it is originally communicated into correct, conventional, and publishable form. This is generally accomplished manually by engraving or lithography.

The use of computer hardware and software has been proposed to speed or simplify these traditional steps. It was the communication problem that was attacked first because it was recognized early that computers had the potential of doing much more than simply printing music. The computer can be a valuable tool also for music composition and performance as well as analysis of musical styles. For all of these purposes, musical compositions must be representable in characteristically linear computer-readable form. In order to communicate a musical composition from one computer to another, or even from one program to another, the two-dimensional form of the music needs to be coded somehow into a one-dimensional sequence of characters. Originally, the communication media were punched cards and magnetic tape. Now, descriptions of musical scores are more likely to be communicated on floppy disks, telephone lines, or coaxial cable, but the sequential nature of communication still requires the one-dimensional encoding of the score.

The most successful work so far on the communication problem has been done by Bauer-Mengelberg et al. [2]. Their language, DARMS, was designed for use as an input language for music-printing systems as well as music-analysis programs. An important contribution of DARMS was the representation of music at an appropriate level of abstraction. For example, in DARMS, one explicitly states the pitch and time of each note, but not the exact Cartesian coordinates of the note head; asking a composer to supply the coordinates of every symbol in the composition would clearly be demanding much more work than required to manually engrave the composition. Furthermore, this information is almost certainly a waste of time and space for a program that does, say, statistical analyses of meter. Unfortunately, by "modern" standards of human–computer interface, DARMS is a very difficult language because it uses numbers and single-character abbreviations rather than words recognizable by musicians.

Another even more difficult language is proposed by Byrd [1] as part of his SMUT system. This language requires the use of numeric codes in fixed columns of punched cards, but this poor human interface is explained by his intention of providing merely an interface between a music-composition program and a music-printing program.

This last consideration raises the possibility that a good solution to the input and output problems might eliminate the need for any serious attention to the communication problem. This, implicitly, is the position taken by those who have worked on the input problem. An example of recent, very good work on the input problem is Mockingbird [7], a system based on an electronic synthesizer and a powerful single-user computer (a Xerox Dorado). Mockingbird allows a composer to enter a composition (for keyboard instruments only) by playing it on the synthesizer keyboard. A staff bearing note heads corresponding to the played keys is displayed on a high-resolution video monitor, and the composer must then manually (but with much software assistance) add durations, beams, bar lines, etc. The resulting composition can then be played back through the synthesizer for evaluation, and printed on a laser printer, yielding a publication-quality score. Mockingbird would be of enormous value to a composer, but even with the low current prices for conventional hardware, Mockingbird, were it sold commercially, would probably go for more than $100,000. Needless to say, few composers could afford such a device. Even if the price were to come down considerably, Mockingbird's lack of solution to the communication problem would remain a handicap. A printed score must be produced at the composer's site, and if it fails to conform to a publisher's requirements as to size or spacing, the composer would have to redo some of the work, produce another paper original, and resubmit it to the publisher. Also, a collaborator at a different site, unless he or she had identical hardware and software, would have to work from the paper version rather than the electronically manipulable form available to the first composer. Although there are a number of simpler, commercially available systems for personal computers that are similar in spirit to Mockingbird, as a rule they suffer from typographical inadequacies such as low-resolution graphics, solely horizontal beams and slurs, and poor note spacing. Perhaps the best of these is Professional Composer [9] for the Apple Macintosh—a system that can produce full orchestral scores, extract parts, and transpose. Its typography, although relatively good, is still not up to the standards of published scores, but even

if this is improved in future versions, the system will still suffer from the device dependency and closed architecture inherent in the Mockingbird model.

The output problem—the problem of producing a printed score from a linear description like DARMS—has been handled with varying success. Because of the tremendous variation in the music notations in use, especially by modern composers, there are inevitably situations where every music-printing program will be inapplicable. Rather than exhibiting occasional defects, however, most programs are limited in other ways that severely limit their utility. SMUT, for example, can handle only a single voice per staff. Mockingbird handles arbitrary chords and several voices per staff, but cannot produce orchestral scores. Characteristic of all music-printing systems is their limited ability to mingle text with music notation: Titles and composers' names can often be printed only in fonts of poor quality compared to the music notation they accompany. Not all music printed by computer is of low quality, however. The music-printing system implemented by Smith [13] produces printed scores of quite high quality, but unfortunately, the system appears to work from an extremely detailed and primitive music-description language.

Despite their drawbacks, these music-printing programs do reflect a great deal of effort and creativity in handling the countless conventions that musicians expect to be followed in printed music. Ross [11] has attempted to enumerate some of these for manual music production, and Gomberg [3] has discussed ways of algorithmically producing the proper effects in music-printing programs. Specifically, Gomberg discusses the algorithms he uses in his program, which prints music from a DARMS description, but the algorithms are applicable to other music-printing systems as well.

As mentioned in the introduction, developments in the field of text processing generally, and particularly mathematical typesetting, have some relevance to the problem of printing music. For example, the text-formatting program called Scribe, developed by Reid [10, 12], embodies a unique and award-winning philosophy in the design of its document-description language. Specifically, the language attempts to mimic, as nearly as possible, the appearance of a manuscript "marked up" by an editor prior to typesetting. The user imagines that a Scribe input file is a typescript that has been marked up by an editor with circles and marginal notes indicating special fonts or other treatments of the text that cannot be accurately reflected in a typescript. To remain within the confines of a computer text file, however, braces, {...}, are substituted for the circles, and the bracketed text is preceded by a command word to replace the marginal note. In Scribe, for example, one can type `@chapter {...}` to indicate that the text within the braces is the title of a chapter. In the final printed document, these words will appear in a form suitable for the title of a chapter, regardless of the capabilities of the output device being used. This deceptively simple philosophy has led to a text-formatting program that is remarkable both for its ease of use and its independence of particular output devices.

A second relevant development in nonmusical typesetting is the program Metafont, written by Knuth [5]. Metafont facilitates the construction of fonts for computer typesetting by simulating the effect of moving a pen along smoothly curving paths through points specified by the font designer. Regions bounded by such paths can be filled in automatically by Metafont, thus defining the shape and weight of a character. The language with which one describes characters to Metafont is very general, and more importantly, with just one such Metafont description a designer can create a large family of fonts differing in size, boldness, slant, etc. Moreover, this takes only a matter of days rather than the months or years it has taken in the past. The benefit of this facility to music printing lies in the fact that in traditional music notation the extreme variability of beams and slurs requires them to be a slightly different shape in nearly every instance; access to a system like Metafont would also allow a composer using modern notation to construct these new symbols quite easily.

The last and most important development is TEX, another computer program designed by Knuth [6]. TEX is the extremely powerful typesetting program alluded to earlier, which excels at mathematical typesetting. Among its significant features are the portability with respect to computers and output devices, the built-in power to handle sophisticated spacing and alignment problems, and its flexibility (in fact, it contains within it a general-purpose programming language). TEX endeavors to solve both the communication and output problems for mathematical typesetting; it has succeeded so well that it has been adopted by the American Mathematical Society as the basis for printing *Mathematical Reviews*, and for the electronic exchange of mathematical manuscripts. For this reason and because of its portability, TEX is currently in use at many universities and at several commercial printers. It is worth noting that, to facilitate the communication of computer-readable mathematical texts, Knuth and the American Mathematical Society, which in part funded TEX's development, have chosen to put the TEX software in the public domain, making it essentially free for any organization or individual who

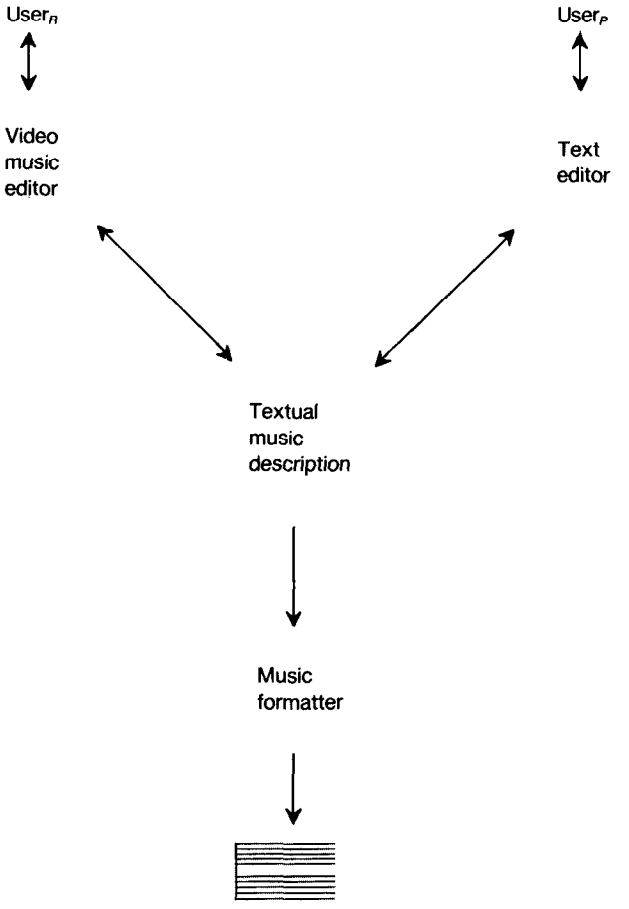
would like to use it. If TEX were to form the basis of a good music-printing system, its wide distribution and portability would make that music-printing system available almost immediately to many university music schools, composers, and publishers.

Project Goals

The primary goal of the music-printing project under way at the Ohio State University is to develop a computer system that produces publication-quality printed music with all the flexibility of good text formatters. Of secondary, but still great, importance are the complementary goals of portability and minimal hardware, which will ensure low cost and encourage wide availability. With current technology, maintaining low cost works to discourage the extensive use of interactive graphics, but not wanting to deny graphics to those who can afford it, we have decided to provide a graphical editor for music notation as an option.

At the center of the basic system (Figure 1) is a music-description language. A description of a piece of music can be created and changed by a user with a small budget ($User_P$) using an arbitrary text editor. Another user with the means for a bit-mapped graphic display and greater computing power ($User_R$) can operate on the same piece of music, but in this case the actual score rather than merely a textual description of it. In both cases, a printed copy of the composition is obtained by passing the textual music description to the music-formatting software.

In this scenario, the electronic communication of musical scores between composers, performers, and publishers will be possible, as will systematic changes to scores, such as extraction of parts and changes of key. Also, the delays and expense associated with proofreading and correcting scores for publication will be minimized. It is easy to envisage the eventual adoption of such a system as a standard by a musicians' organization in very much the same way as the American Mathematical Society has adopted TEX.



FEATURES OF THE MUSIC-DESCRIPTION LANGUAGE

At Ohio State, the music-printing project (called the MusiCopy Project) has progressed to the point where we have defined a music-description language that is ready for implementation and testing. This attempt to capture musical scores in a textual form has led to a language with a number of novel features from both the computer science and music points of view.

An Abstraction

The importance of the music-description language in this project cannot be overemphasized. The presence of an ordinary text file representing a printed piece of music facilitates the electronic communication of scores between dissimilar computers and between users with unsophisticated equipment. More importantly, though, the music-description language provides an abstraction of the ultimate printed music that is very important to formatting and communication.

The primary characteristic of a correct abstraction for document formatting, whether textual or musical, is that objects be coded according to their function in the document rather than their position in the document. With this kind of coding, we are able to insert or delete sections of a document and have the remaining contents move around automatically on the page and from one page to another to properly compensate for the changes. In a textual document, this requires that the software know that certain pieces of text are section and chapter headings so that they can be treated differently from the running text of the document. This kind of functional coding also allows us to change our minds about page size or the number of columns and expect the text to be rearranged accordingly.

Beyond these general kinds of changes, there are special kinds of changes that are unique to music printing. A tremendously time-consuming activity for music copyists is the extraction of parts, in effect producing two equivalent copies in different formats for every piece of music they prepare. One copy of the piece, for conductors and scholars, simultaneously shows the performances of all the instruments used in the piece. The other copy, for performers, consists of separate sheets of music for each performer. Another frequent systematic change is transposition, the raising or lowering of all the notes by the same amount. This is part of the extraction of parts for certain "transposing" instruments, and it can be done to accommodate singers whose vocal ranges differ from the range required by the original piece. Another use of computer-readable music descriptions that is less critical, but was nonetheless kept in mind in the design of the language, is mechanical performance. Although there is no intention in the near future to try to replace orchestras with computers, it is hard to imagine a better way to "proofread" a score than to let its composer hear an approximation of its ultimate human performance. Extraction of parts, transposition, and performance are systematic changes that can be done mechanically only if the representation of the score contains explicit information about performers, keys, and pitches.

Therefore, a good document-description language is a necessity for a good computerized text-formatting system, and a good music-description language is required for flexible computer printing of music. This is not to deny the obvious benefit of immediate visual feedback on the appearance of a book or musical score. As was already pointed out, a visual music editor is a planned part of the music-printing project, but the music-description language must be complemented, rather than replaced, by the elegant input system.

Other Language Features

Declarative Rather Than Procedural. The declarative versus procedural debate is often heard concerning programming languages, but rarely concerning other kinds of computer languages. Music descriptions in most, if not all, previous music-description languages must be understood as incremental sequences of instructions much like computer programs. Understanding what a particular instruction means in the middle of such a music description requires mentally interpreting the entire sequence preceding the instruction in question. Such procedural languages are easy to implement, but are hard to understand. By contrast, our new declarative language describes the music as a static object in a hierarchical fashion, by assembling several notes into a measure, and then assembling several such measures into a musical phrase, for example. In such a language, every syntactic expression has a well-defined meaning as a smaller or larger musical object, and the amount of contextual information required for a complete understanding of the expression is very small—comparable to the depth of the hierarchy rather than the length of the description. This is exactly the Scribe philosophy, but extended to music description.

One desirable consequence of the declarative philosophy of the language is its block structure. In the mind of the musician, and in the music-description language, a piece of music consists of a sequence of musical phrases (*groups* in the music-description language), each of which is composed of several

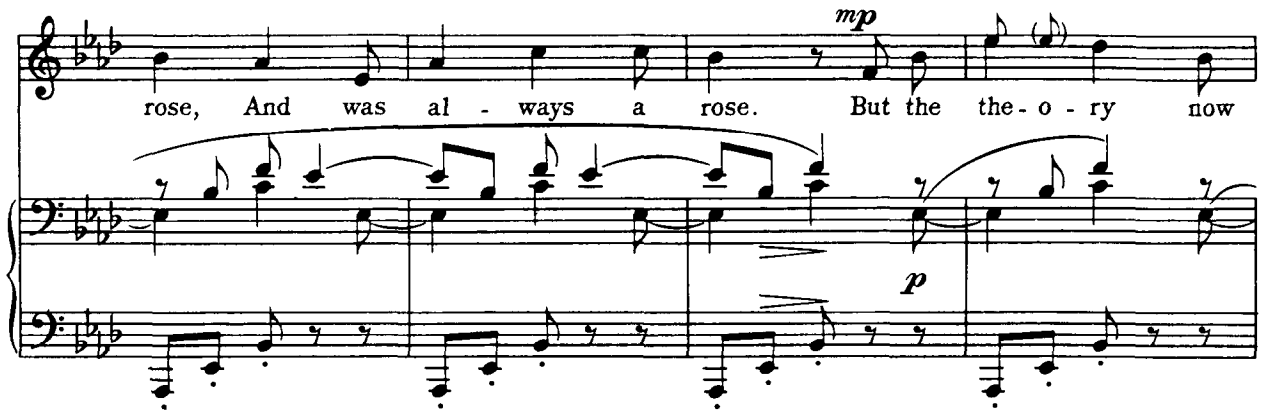


FIGURE 2. A Typical Musical Phrase

shorter phrases, down to individual notes or chords. One such musical phrase might be a series of "long grace notes," small notes that require different typography than the ordinary notes of the piece. The appropriate syntax to indicate this would be `longgrace (...)` where the description of the grace notes is enclosed in parentheses preceded by the keyword `longgrace`. Everything between the parentheses is typeset as grace notes, and upon completion of the grace notes, the notational conventions that prevailed before the grace notes began are restored. This kind of group can appear in larger groups and can contain smaller groups. A reasonable example of a contained group in this case would be a beamed group of notes (i.e., notes, such as 8th or 16th notes, connected by bars to indicate their duration). These nested groups save the musician the trouble of remembering to do and undo all the changes that might be required to set grace notes or beams in a more procedural language.

Two-Dimensional. Respecting the fact that printed music is two-dimensional, the music-description language allows the copyist to fill in the parts of the score both horizontally or vertically. The line of music shown in Figure 2, like lines from most scores, consists of several staves that are bound together vertically by a bar at their left end, and that continue horizontally across the page. The horizontal dimension symbolizes the progression of time; the symbols lined up vertically represent events that take place simultaneously. In what order should the notes of this piece be recorded in a textual music description? The answer is that the best order depends on the musical context. Therefore, our music-description language allows the music copyist to choose one of two modes of entry, row by row or column by column (where a column is one measure wide), and to switch from one to the other at arbitrary points in the piece.

In the case of the musical phrase shown in Figure 2, the top staff contains the notes that are to be sung (and conceptually the words themselves), the second staff represents the notes to be played by the pianist's right hand, and the third staff the notes to be played by the left hand. Assuming these were declared to have the names `words`, `right`, and `left` at the beginning of the description, the following two forms would be equivalent ways of entering the first two measures of this line:

```
block
  [words (measure(...) measure(...))
  right (measure(...) measure(...))
  left (measure(...) measure(...))]

block
  [measure (words(...) right(...)
    left(...))
  measure (words(...) right(...)
    left(...))]
```

The first of these forms fills in the measures row by row while the second fills them in column by column. (In both examples, the use of brackets is optional, and the order of the staff names is arbitrary.)

Expressive of Concurrency. From the point of view of programming-language design, the music-description language is unusual in that it must allow for improperly nested blocks to respect the fact that in music many things go on simultaneously, and the times at which they start and stop can be interleaved arbitrarily. Slurs, for example, frequently extend from the middle of one measure to the middle of the next, a situation that can be described as follows:

```
measure (... begin [slur] ...)
  measure (... end [slur] ...)
```

In this example, **measure (...)** defines a block, as does **begin [slur] ... end [slur]**. The fact that the first measure ends before the slur that began in it ends is no problem in music and therefore must be within the capabilities of the music-description language.

Expressive of State Changes. Despite the advantages of an entirely block-structured language with nicely nested scopes, other features of music printing force us to deviate from this model. As a group, these features have come to be called *changes* (not to be confused with changes in jazz), which are best characterized by changes of key signature. When a new key signature appears in the middle of a score, it is understood that it replaces the previous key signature permanently, not temporarily as it might if it signaled entry into a nested block. Notationally speaking, one does not ever return to the old key signature. Rather, one replaces the new key signature with yet another that might happen to be the same as the original. For changes of this sort, the music-description language provides commands that cause permanent changes to the interpretation of the music description.

A special case of this kind of state change is a change of meter. The convention in music is that meter changes must occur simultaneously for an entire orchestra, not just for one instrument. The question of how to handle this syntactically was resolved by appealing to a protocol that might be useful in a graphical music editor. If a copyist were editing a score visible on the screen of a computer workstation, it would seem reasonable to select a meter signature from a palette at the periphery of the screen and drag it to a point on the score. When released, one would expect it to insert itself there and replicate itself at the same point in time in all the other staves. An analogous behavior was adopted for the music-description language, where such a global change is syntactically similar to a local change (e.g., a key change), but its effect propagates to all staves, regardless of the staff in which it falls.

Verbose, But with Abbreviations. The music-description language is verbose in the sense that it uses long, meaningful words rather than short abbreviations to indicate the structure of the music. The intention is that a music copyist using the system should find many of the keywords meaningful and need to learn as little new vocabulary as possible.

At the same time, the language allows users to design their own system of abbreviations by declaring them at the beginning of the music description. For experienced users, this can compensate for the verbosity built into the basic vocabulary of the language. More important, like subroutines in a typical programming language, the abbreviation mechanism can be used to clarify and make more accurate the use of repeated elements in a piece of music. If a chord, melody, or rhythm is used often throughout a piece, the user will be able to give it a name at the beginning of the music description and then use that name repeatedly throughout the piece. For example, in a piece with many occurrences of four consecutive quarter notes of the same pitch, the music description can be made easier to read and shorter by including the definition

```
define quarters pitch
  ((4;pitch) (4;pitch)
  (4;pitch) (4;pitch))
```

at the beginning of the piece and by using **quarters C4** to get four middle Cs and **quarters D4** to get four Ds. This is handled by a macroprocessor; **quarters** is the name of a macro with one parameter called **pitch**, and **quarters C4** expands into **(4;C4) (4;C4) (4;C4) (4;C4)**.

High-Level with Escapes. The language is designed to allow music descriptions at the same high level of abstraction used by musicians copying or studying scores. That is, a piece of music is described in terms of pitches, durations, slurs, and crescendos, etc., and not in terms of x–y coordinates, spacing, and beam or slur shape. It is the music-printing software that chooses shapes and spacings that properly represent the concepts in the musician's mind. This is admittedly a heavy burden to place on the software, which cannot be expected to replace or replicate the aesthetic judgement of an experienced music copyist. The software will make occasional mistakes, and therefore escapes from the high-level abstraction to the details of typography are possible. For example, **beam (...)** will usually suffice to create a beamed group of notes, but in the rare case where the beam is placed awkwardly, one may enter something like **beam 2–3 (...)**, where the numbers indicate the vertical positions on the staff at which the beam should begin and end.

TEXT-FORMATTING REQUIREMENTS

The immediate plan for the music-printing project is to prototype the system around TEX, even though neither TEX nor any other currently available text-formatting system is very well suited to all the problems of music printing. For the prototype, the special problems of music can be solved most rapidly by writing pre- and postprocessors for TEX, although a

long-range goal is to develop more general text-formatting software to underlie the music-printing system.

An obvious requirement for an underlying formatter is that it have the full capabilities of a good text formatter. Published music almost invariably contains a significant amount of ordinary text, not only in titles and lyrics, but also in tables of contents, commentaries, and other front matter. A music textbook, with small segments of music intermingled with paragraphs of text, should not be beyond the capabilities of the music-printing system.

A feature of special importance to both text and music typesetting is the ability to find optimal line and page breaks. Most text formatters simply put as many words as they can on a line before moving on to the next, possibly creating uneven word spacing from one line to the next. TEX attempts to find a series of line-break points that minimizes the need to stretch word spaces over the entire paragraph. In text, this results in noticeably better paragraphs. In music, where it is customary to "cast off" the measures of a piece into full lines, with no short line at the end of a piece, optimal line breaking is mandatory. Sometimes music printers even go a step further and cast off a score to fill some number of whole pages. Not even TEX will perform optimal page breaks in a way that permits this kind of formatting in one try, although the TEX algorithm is better than that used by other text-formatting programs. Excellent results have been obtained experimentally by Plass [8] using TEX's optimal line-breaking algorithm to find page as well as line breaks; with an algorithm like this, music could be cast off into whole pages in one try.

More specific to music formatting is the ability to superimpose text on a background. In music, the staff lines and perhaps the clefs and bracketing at the left margin are best thought of as a background upon which the music is printed. The problem, however, is more complicated than merely overstriking the music on a fixed form because the background of staves must conform in ways that may vary from page to page. The first page, for example, might require an indentation of the first line to make room for the names of instruments, and the last page may not have a full complement of lines. Anywhere in the piece, extra space may be needed between staves to make room for conflicting notation. Roughly speaking, what is needed is a kind of background that can stretch and break to a new line following the layout of an arbitrary foreground. An incidental benefit of such a feature in a text formatter would be a convenient mechanism for producing struck-out text in legal and legislative documents.

The musical features that have probably been the greatest impediment to mechanizing music printing are the extended beams and slurs. Because the spacing of notes is so flexible in music, a fair premise in designing a music formatter is that no two beams or slurs are alike. The rules for choosing the positions and shapes of these figures are beyond the scope of this paper, but given that their parameters can be determined, we need a formatter that is capable of drawing smooth curves and straight lines as if with pens. A graphics preprocessor like the PIC preprocessor for TROFF [4] would not be suitable because the parameters of the graphics will depend critically on decisions made later in the formatter. A more integrated approach is necessary, which would pay off not only in music printing, but also in the printing of calligraphic writing systems like Arabic.

A last feature expected of the ideal music formatter is a sophisticated mechanism for aligning and spacing tables. In a measure of music, the notes that begin on the same beat are aligned vertically and are followed by space that is roughly proportional to their duration. Seen in this way, constructing a measure of music is very much like constructing a table, where the columns correspond to the beats at which the notes begin, and column widths are figured from the notes' durations. Complexity arises, however, when it is not always simple notes that appear in these columns. Some notes have sharps or flats preceding them or dots following them, but should nevertheless be aligned on the center of their note heads. Words that are to be sung are aligned somewhat left of their notes. Ideally, these protrusions to the left or the right of the note heads should not take space of their own, but should occupy the space present due to the durations of their note and the previous note. If the duration space does not provide enough room, however, the note spacing must be opened up to avoid overstriking. TEX's primitives for table building do seem to allow this kind of complexity, but only with great effort. A challenge for the next generation of text formatters is providing simple primitives that allow this kind of alignment.

CONCLUSION

The foregoing paper has discussed a language whose primary purpose is the description of musical scores for the purpose of high-quality printing. The appendix that follows defines in detail the current version of the language. Implementation of a music-printing system based on this language is proceeding, but even in the absence of an implementation, it is hoped that the emphasis here on appropriate abstraction, communication, and device independence has been useful to others interested in all kinds of computer languages.

APPENDIX: DEFINITION OF THE LANGUAGE

As is customary in defining a computer language, the sequence of characters that is input to the music-printing system is broken into a sequence of relatively short *tokens*. Roughly speaking, the tokens are the smallest meaningful elements of the language and are analogous to the words of a natural language. The rules for recognizing the tokens of the music-description language are given below in the section headed "Microsyntax." The rules of syntax by which tokens are assembled into phrases, and phrases assembled into larger phrases to ultimately obtain a music description, are discussed under "Syntax and Semantics." The meanings of the various components of a music description are given informally in the paragraphs immediately following the syntax of the components.

The syntax of the music-description language is given in a fairly standard form of BNF. Terminal symbols, including keywords and punctuation marks, are printed in a typewriterlike font, for example, **row**, **(**, and **;**. Nonterminal symbols are printed in roman and enclosed in angle brackets. A superscript plus ($^+$) indicates that the previous symbol or braced expression may appear one or more times; a superscript asterisk (*) means that the previous expression may appear zero or more times. Braces may also enclose a vertical stack of expressions, indicating that any one of the items in the stack may be chosen. Square brackets enclosing an expression mean that the expression is optional, and, more generally, those enclosing a stack of expressions indicate that any one or none of the items in the stack may be chosen.

Microsyntax

The tokens of a music description are *words*, *numbers*, *tests*, and *punctuation marks*. A word is a sequence of letters that is always the longest possible sequence that does not contain nonalphabetic characters. There are several special kinds of word tokens that are distinguished by their different uses in the grammar. The broadest group of words is the *names* (e.g., staff names and voice names). Any word may be used as a name, and this choice is indicated in the grammar with the symbol ⟨word⟩. Another group of words are *keywords*, which mark the beginning of specific parts of a music description. These words (e.g., **system** and **staff**) are written out explicitly in the grammar in a typewriterlike type. Three more categories of words are collectively known as *note names*, and called ⟨simple note⟩, ⟨discretionary accidental⟩, and ⟨mandatory accidental⟩ in the grammar. The syntax of these note names is given in the following short grammar, and their meanings are described in connection with their use in the next section:

⟨simple note⟩ →
  A | B | C | D | E | F | G | a | | c | d | e | f | g
⟨discretionary accidental⟩ →
  ⟨simple note⟩⟨discretionary tail⟩
⟨mandatory accidental⟩ →
  ⟨simple note⟩⟨mandatory tail⟩
⟨discretionary tail⟩ →
  ff | f | n | s | ss
⟨mandatory tail⟩ →
  FF | F | N | S | SS

A number is a sequence of digits, possibly containing a decimal point, and also the longest possible such sequence of digits. Number tokens are indicated in the grammar as ⟨number⟩ and have the following microsyntax:

⟨integer⟩ → ⟨digit⟩$^+$
⟨number⟩ → ⟨integer⟩
  | ⟨integer⟩ . ⟨integer⟩
  | . ⟨integer⟩
⟨digit⟩ → **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**

A text is a sequence of characters enclosed in double quotes. Texts are used to introduce words for titles, lyrics, dynamics, etc., and to provide an escape from the music-description language to the underlying text-formatting system upon which the music-printing system is built. The latter facility is for the purpose of selecting characters from special fonts, doing unusual positioning, etc. If it is necessary to use a double quote within a text, it must be represented by two consecutive double quotes; in the grammar, these are denoted by ⟨text⟩.

Punctuation-mark tokens include any of the printable, nonalphanumeric characters in the ASCII character set, except for decimal points within ⟨number⟩s and percent signs. In the grammar, these are shown explicitly in the typewriter font.

Spaces, tabs, new lines, and comments may appear anywhere between tokens. A percent sign marks the beginning of a comment indicating that the percent sign and the rest of the line it is on are ignored.

In the grammar given below, there are many rules that specify the use of balanced parentheses around a series of simpler components. In any

such case, the balanced parentheses, (...), may be replaced by balanced square brackets, [...], balanced angle brackets, ⟨...⟩, or balanced braces, {...}. In cases where the left parenthesis is preceded by a word, for example, **row(...)**, the alternative construction **begin(row)** ... **end(row)** may be used.

Macros

Although the music-description language is verbose by design, it has a facility for defining and using abbreviations, which takes the form of a simple macro language. Formally, macro definitions and uses have the following syntax:

⟨macro definition⟩ →
  define⟨macro name⟩⟨parameter name⟩*
    ⟨balanced token list⟩
⟨macro use⟩ →
  ⟨macro name⟩⟨parameter⟩*
⟨macro name⟩ → ⟨word⟩
⟨parameter name⟩ → ⟨word⟩
⟨parameter⟩ →
  ⟨nonparenthesis token⟩
  | ⟨balanced token list⟩
⟨balanced token list⟩ →
  ([⟨nonparenthesis token⟩ / ⟨balanced token list⟩]*)

A macro definition is initiated by the keyword **define** followed by a name for the macro. After the macro name is an optional list of names to be used to indicate parameter substitutions. Following this is the list of tokens into which the macro should expand. This list is enclosed in parentheses, and all parentheses within it must be balanced.

A macro is used, or expanded, whenever the ⟨macro name⟩ token is encountered after its definition. (Strictly speaking, it is expanded only when it is found outside a macro definition or parameter.) If the macro definition included no ⟨parameter name⟩s, then the expansion token list simply replaces the macro name. More generally, if the macro definition has *n* ⟨parameter name⟩s, then an expansion replaces the macro name and *n* additional tokens or parenthesized token lists. The expansion is constructed by scanning the expansion token list for occurrences of ⟨parameter name⟩ tokens and replacing them with the corresponding parameter token or token list. In both kinds of replacements, whether replacing a parameter name or a macro name, one set of enclosing parentheses is stripped off before the replacement is made. After a macro name and its parameters have been replaced, scanning for macros begins anew at the beginning of the replacement, allowing macros to use or define other macros.

This macro facility could be enhanced to include conditional expansion and nested scopes if its use in the music-description language warrants it.

Syntax and Semantics

In the following definition of the music-description language, the language is described in small pieces; for each piece of the language, its syntax is given first as a small number of BNF rules, and the meanings of the syntactic objects thus defined are described immediately afterward in a paragraph or two.

⟨piece⟩ → ⟨heading⟩⟨section⟩+

⟨heading⟩ → [**title** ⟨text⟩ / **poet** ⟨text⟩ / **composer** ⟨text⟩]*

⟨section⟩ → ⟨system⟩⟨block⟩+

The music-printing system deals with one ⟨piece⟩ on each run. A ⟨piece⟩ might be a full piece of music, but it also might be a single movement. The criterion for what constitutes a ⟨piece⟩ is that it should begin with a title of some sort at the top of a page. The title and any other textual material that are to appear at the top of the first page are constructed in the ⟨heading⟩. The ⟨text⟩ associated with **title** is centered at the top of the page, and the **poet** and **composer** ⟨text⟩s are below it, left and right adjusted, respectively. A ⟨section⟩ is a portion of the score printed on a uniform system of staves. A piece of music that is to be printed without any changes of system may be written in a single ⟨section⟩; ⟨system⟩ defines the characteristics of the system on which the ⟨section⟩ is to be printed.

⟨system⟩ → **system**(⟨bracket⟩+)
⟨bracket⟩ → **bracket**{⟨brace⟩+}
  | ⟨brace⟩
⟨brace⟩ → **brace**⟨print names⟩(⟨staff⟩+)
  | ⟨staff⟩

Basically, a system consists of a number of staves. The staves, however, may be joined at the left margin into large groups with brackets and into smaller groups with braces. Therefore,

⟨system⟩ is described as a series of ⟨bracket⟩s, each of which is described as a series of ⟨brace⟩s, each of which, in turn, is described as a series of ⟨staff⟩s. A ⟨bracket⟩ or ⟨brace⟩ written without the keywords **bracket** or **brace** is printed without any bracketing symbol at the left of the line. In spite of the order in which things are specified in the music-description language, braces are printed to the left of brackets. All the staves of a system are connected by a bar line at their left ends. Additional bar lines have exactly the same extent as ⟨bracket⟩s. The symbol ⟨print names⟩ represents names to appear to the left of a braced group of staves, a full name for the first line of the piece and an abbreviation for subsequent lines.

⟨staff⟩ →
  staff⟨staff name⟩
    ⟨print names⟩⟨format⟩⟨voice⟩*
  | **tempo**
⟨voice⟩ → **voice**⟨voice name⟩
⟨staff name⟩ → ⟨word⟩
⟨voice name⟩ → ⟨word⟩

Every ⟨staff⟩ has a ⟨staff name⟩ that is used to refer to the staff from other parts of the music description. The ⟨staff name⟩ is also automatically the name of a voice on the staff, which may be used in addition to, or in the absence of, any explicit ⟨voice⟩s in the staff description. Like a braced group of staves, a ⟨staff⟩ has optional ⟨print names⟩ that are printed to the left of each line of the score. A ⟨staff⟩ written simply as the keyword **tempo** marks one of perhaps several positions in the system at which tempo and rehearsal marks are to be placed. These spaces in the system can be thought of as staves with no lines; things can be placed on all of them simultaneously by referring to the special ⟨staff name⟩ **tempo**. If there are no explicit **tempo** staves in the system definition, one is assumed to exist above the first staff in the system.

$$\langle\text{format}\rangle \rightarrow \left[\begin{array}{l}\textbf{size}\ \langle\text{number}\rangle \\ \textbf{topspace}\ \langle\text{number}\rangle \\ \textbf{bottomspace}\ \langle\text{number}\rangle \\ \textbf{lines}\ \langle\text{number}\rangle\end{array}\right]^*$$

Using ⟨format⟩ in a ⟨staff⟩ definition allows the user to control the appearance of a staff. The keyword **size** is followed by a numeric staff size, the largest being 0, the smallest being 8, and default is 3. The keywords **topspace** and **bottomspace** specify the amount of white space above and below the staff. The space is given in units equal to the staff line spacing for the chosen staff size. The default for both **topspace** and **bottomspace** is 3. The keyword **lines** allows the copyist to specify the number of lines in the staff; the default, of course, is 5. The number of lines may be zero, in which case the staff is invisible on the page. It will, however, occupy space, and notation may be placed on it. This option is intended for lyrics and perhaps for dynamics in the center of a grand staff. It is also used implicitly for the special **tempo** "staves."

$$\langle\text{print names}\rangle \rightarrow \left[\begin{array}{l}\textbf{fullname}\ \langle\text{text}\rangle \\ \textbf{abbreviation}\ \langle\text{text}\rangle\end{array}\right]^*$$

The names of staves and braced groups that appear in the left margin of the printed score are specified as ⟨print names⟩. The **fullname** is printed on the first line of the score, and the **abbreviation** is printed on all subsequent lines. If either one or both are omitted, the corresponding places on the score are left empty.

⟨block⟩ → **block**(⟨row⟩⁺)
  | **block**(⟨column⟩⁺)

A ⟨block⟩ is a rectangular chunk of score that includes all the staves of the system, vertically, and one or more measures, horizontally. The measures of a ⟨block⟩ may be filled in either row by row or column by column, whichever is more appropriate musically.

⟨row⟩ →
  ⟨voice name⟩(⟨measure entry⟩⁺)
⟨measure entry⟩ →
  measure(⟨measure⟩)

If a ⟨block⟩ is to be filled in row by row, the rows, which correspond to voices, can be supplied in any order. Each ⟨row⟩ begins with a ⟨voice name⟩; following the ⟨voice name⟩ are the ⟨measure⟩s that belong to that voice in left-to-right order. The ⟨row⟩ with the largest number of measures defines the width of the ⟨block⟩, and any ⟨row⟩s that have fewer ⟨measure⟩s are given trailing empty measures to make all ⟨row⟩s the same length in the ⟨block⟩.

⟨column⟩ → **measure**(⟨voice entry⟩⁺)
⟨voice entry⟩ → ⟨voice name⟩(⟨measure⟩)

If a ⟨block⟩ is to be filled in column by column, the columns must be supplied in left-to-right order. Within a column, which is always one measure wide, the measures may be supplied in any

order. Each ⟨measure⟩ with a ⟨column⟩ is prefixed by a ⟨voice name⟩, and any voices that are not mentioned are left empty in that column. The width of the ⟨block⟩ is the number of ⟨column⟩s in the block.

⟨measure⟩ → ⟨group⟩⁺
⟨group⟩ → ⟨chord⟩
| ⟨local change⟩
| ⟨global change⟩
| ⟨group head⟩⁺(⟨group⟩⁺)

A ⟨measure⟩ in its simplest form is just a series of notes, chords, and rests, all of which are instances of the nonterminal ⟨chord⟩, which is described below. The total duration of these notes, chords, and rests should add up to the duration of a measure. The music-printing system will warn users about violations of this rule, but will not prevent the music from being set. In addition to notes, some measures will contain changes of key signature, clef, etc., which typically occupy space in the measure, but do not contribute to the duration of the measure. These changes are instances of the nonterminals ⟨local change⟩ or ⟨global change⟩, depending on whether their effect is over a single staff or the entire system. The third kind of object that can occur in a ⟨measure⟩ is a ⟨group⟩, which itself most likely contains several notes. A ⟨group⟩ describes a series of notes that are grouped together musically, for example, with beams or slurs. The ⟨group head⟩ indicates the kind of group, and because notes may be both beamed and slurred, among other things, it is possible to attach several ⟨group head⟩s to one ⟨group⟩.

⟨group head⟩ → **longgrace**
| **shortgrace**
| **multiple**
⟨number⟩:⟨number⟩
| **stemsup**
| **stemsdown**
| **stemsboth**
| **beam**
[⟨beam shape⟩]
| **primarybeam**
[⟨beam shape⟩]
| **slur**
[⟨slur shape⟩]
| **tie**
[⟨slur shape⟩]
| **cresc**
[⟨staff degree⟩]
| **dim**
[⟨staff degree⟩]
| **overlay**
⟨overlay shape⟩
⟨overlay shape⟩ → [**start**⟨text⟩]
[**line** | **dots** | **trill**]
[**stop**⟨text⟩]
[⟨staff degree⟩]
⟨beam shape⟩ → ⟨staff degree⟩ −
⟨staff degree⟩
⟨slur shape⟩ → [⟨staff degree⟩ −]
*⟨staff degree⟩

A ⟨group head⟩ can indicate—with the keywords **longgrace** and **shortgrace**—whether the notes enclosed in the ⟨group⟩ are long or short grace notes. In either of these cases, the notes of the group are set in the proper small size according to their durations, but the full grace note group contributes no duration to the enclosing ⟨group⟩ or ⟨measure⟩.

A ⟨group⟩ headed by **multiple** occupies a duration reduced by the ratio following the keyword. In addition, the notes of the group are bracketed, if necessary, and the first number of the ratio is printed with the group. For example, `multiple 3:2` defines a triplet. The total duration of the notes within a **multiple** group must be a multiple of the first note of the ratio.

The keywords **stemsup**, **stemsdown**, and **stemsboth** indicate the direction in which stems should point in the group. The default state of affairs is **stemsboth**, indicating that stem directions depend on the staff degree of the note heads. These can be nested, so that the copyist can create a temporary region of **stemsboth** within a larger region of **stemsup**.

The keyword **beam** means that the grouped notes should be beamed together, where the number of beams is determined by the durations of the notes within the group. Likewise, the keyword **slur** indicates that the notes should be enclosed in a slur symbol. A group beginning with **tie** is similar to **slur** except that all chords within it must be the same and that the notation produced follows the convention for **ties**.

The keyword **primarybeam** is used to enclose several **beam** groups, indicating that the primary beam of the several groups should continue from one group to the next. If **primarybeam** has a ⟨beam shape⟩, the enclosed **beams** may not have one; their shape is forced by the primary beam.

The keywords **cresc** and **dim** construct the appropriately extended wedge-shaped symbols that will span the grouped notes.

The keyword **overlay** introduces any other extended dynamic or tempo mark. The ⟨overlay shape⟩ allows the composer to fully define the symbol by giving starting text, ending text, and the type of leader with which to connect them.

For **cresc**, **dim**, and **overlay**, the figure is placed under the staff unless its position is indicated explicitly by ⟨staff degree⟩.

The optional ⟨beam shape⟩ and ⟨slur shape⟩ fields exist to allow the copyist to force beams, slurs, and ties into shapes or positions that differ from the ones the music-printing system would choose. A ⟨beam shape⟩ consists of two ⟨staff degree⟩s, which are taken to be the left- and right-end points of the beam farthest from the note heads. The beam will connect these two points, and the stems will stretch or shrink as necessary to reach the beam. A ⟨slur shape⟩ consists of a series of one or more ⟨staff degree⟩s, which are taken to be evenly spaced points through which the slur or tie should pass.

⟨chord⟩ → **(**⟨duration⟩;
[⟨text⟩⟨staff degree⟩ / ⟨pitch⟩]*
[;{⟨text⟩⟨staff degree⟩}+**)**
| **(**⟨duration⟩**)**
| **(**⟨duration⟩**null)**
⟨duration⟩ → ⟨integer⟩.*
⟨staff degree⟩ → ⟨number⟩

A ⟨chord⟩ may represent more than its name implies. In its full generality, a ⟨chord⟩ defines a vertical stack of symbols that may or may not be connected by a stem. The information after the first **;** lists the symbols (and their staff positions) to be connected by a stem; a stem, however, is only present if it is appropriate for the ⟨duration⟩. This field is used for note heads, primarily, and the notation of ⟨pitch⟩ is provided to simplify this. The information after the second **;** lists additional symbols that are to be present, but not connected to the stem. This field is used for accents and lyrics. Two special cases have their own syntax—rests and null rests. A ⟨chord⟩ consisting only of a ⟨duration⟩ in parentheses generates a rest, whereas the form containing the word **null** generates nothing but the right amount of empty space for the given duration.

A ⟨duration⟩ may be zero or any power of two, followed by any number of dots (periods). A ⟨chord⟩ with a duration of zero contributes nothing to the duration of the enclosing group, and dots in this context change nothing. A nonzero duration indicates the kind of note head to be printed and the presence or absence of a stem: 1 for a whole note, 2 for a half note, 4 for a quarter note, etc. Dots after these duration values add to the note's duration in the conventional way.

A ⟨staff degree⟩ is a number indicating the number of staff spaces above or below the center line of the staff, at which the associated symbol should appear. Positive numbers mean above, and negative numbers mean below. If the staff has an even number of lines, the "center" line is the lower of the two central lines. If the staff has no lines, distances are measured from the center of the space occupied by the staff.

⟨local change⟩ → **key** { ⟨number⟩ **sharps** / ⟨number⟩ **flats** / {⟨mandatory accidental⟩ ⟨register⟩}+ }
| **clef** { **trebleclef** / **bassclef** / **altoclef** / **tenorclef** / **percussionclef** }
| **begin**⟨group name⟩ (⟨group head⟩)|
| **end**⟨group name⟩
⟨group name⟩ → ⟨word⟩

A ⟨local change⟩ changes the key signature or the clef of the current staff, or begins or ends a ⟨group⟩ in the current voice. The latter option allows groups to overlap without being nested, and to cross bar lines.

A key signature may be either a standard pattern of sharps or flats, in which case one of the first two choices is used, or it may be any pattern of sharps, flats, and naturals, in which case they are given as an explicit list of pitches. A clef must be specified before a key signature. Any one of the five standard clefs may be chosen by name. This may someday become a special case of a more general mechanism.

Note that a mismatched group, where, for example, a slur begins in one measure and ends in the next, is treated syntactically as a pair of ⟨local change⟩s, the first a **begin** and the second an **end**.

⟨global change⟩ →
meter⟨number⟩/⟨number⟩

| **barline** { **double** | **final** | **leftrepeat** | **rightrepeat** | **rightleftrepeat** | **dotted** }

A ⟨global change⟩ may appear in any staff, but its effect is to make a change to all the staves in the system. The keyword **meter** introduces a new meter signature across the entire system and must be followed by the two numbers of the signature separated by a slash; the second number of the pair must be a power of two. A **barline** occurring at the beginning or end of a measure changes the bar line at that point across the whole system from the standard single line to one of the more complex symbols. If **barline** appears in the middle of a measure, an extra bar line of the chosen form appears in every staff.

⟨pitch⟩ → ⟨note name⟩[⟨register⟩]
⟨note name⟩ → ⟨simple note⟩
| ⟨discretionary accidental⟩
| ⟨mandatory accidental⟩
⟨register⟩ → ⟨number⟩

A ⟨pitch⟩ that is a ⟨mandatory accidental⟩ is printed as a note with an accidental sign, regardless of whether or not the accidental would be conventionally required. A ⟨pitch⟩ that is a ⟨discretionary accidental⟩ is printed with or without an accidental sign according to the convention that accidentals influence the pitch of all notes of the same register until the end of the measure. A ⟨simple note⟩ is printed without an accidental and is interpreted as the proper pitch according to the same convention. The three kinds of note names are tokens of the language, and they are described syntactically in the section on microsyntax.

A ⟨register⟩ may be any number from **0** through **9**, where **C4** refers to middle C. If the ⟨register⟩ is omitted from a ⟨pitch⟩, and the ⟨pitch⟩ is alone or the first note in a ⟨chord⟩, the register number is inferred to be the one that creates the smallest diatonic interval between the current note and the first note of the previous ⟨chord⟩. If the ⟨pitch⟩ is the second or subsequent ⟨pitch⟩ in a ⟨chord⟩, then the ⟨register⟩ is chosen to create the smallest interval with the previous note in the same ⟨chord⟩. The ⟨register⟩s may be omitted from several successive ⟨chord⟩s, but the first ⟨chord⟩ of any ⟨row⟩ or ⟨voice entry⟩must have explicit ⟨register⟩s.

Acknowledgments. I would like to thank Dean Roush for the musical expertise, and The Ohio State University and the National Science Foundation for the financial means to undertake this project.

REFERENCES

1. Byrd, D. A system for music printing by computer. *Comput. Hum. 8* (May 1974). A description of the SMUT music-printing system.
2. Ericson, R.F. The DARMS project: A status report. *Comput. Hum. 9* (Nov. 1975). A discussion of the design and potential uses of the DARMS music-description language.
3. Gomberg, D.A. A computer-oriented system for music printing. *Comput. Hum. 11* (Mar. 1977). A discussion of the problems of representing the conventions of musical notation in computer algorithms.
4. Kernighan, B. PIC—A language for typesetting graphics. *Softw. Pract. Exper. 12*, 1 (Jan. 1982). A discussion of a "graphics-description language."
5. Knuth, D.E. *TEX and Metafont*. Digital Press, Bedford, Mass., 1979. A broad discussion of computer typography, including somewhat obsolete sections on TEX and Metafont. The current version of TEX is documented in *The TEXbook* [6], and documentation for the recently released version of Metafont is still in preparation.
6. Knuth, D.E. *The TEXbook*. Addison-Wesley, Reading, Mass., 1984. Documentation for the TEX text-formatting system.
7. Maxwell, J.T., and Ornstein, S.M. Mockingbird: A composer's amanuensis. *Byte 9* (Jan. 1984). A discussion of an interactive and graphical computer system for music composition.
8. Plass, M.F. Optimal pagination techniques for automatic typesetting systems. Tech. Rep. CS870, Computer Science Dept., Stanford Univ., Calif., 1981. A study of algorithms for automatically breaking documents into aesthetically pleasing pages.
9. *Professional Composer*. Mark of the Unicorn, Cambridge, Mass., 1985. A computer program for the Apple Macintosh that uses interactive graphics for preparing musical scores.
10. Reid, B.K. Scribe: A document specification language and its compiler. Tech. Rep. CMU-CS-81-100, Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, Pa., 1980. A discussion of the philosophy behind the Scribe text-formatting system.
11. Ross, T. *The Art of Music Engraving and Processing*. Hansen Books, 1970. A detailed tutorial on the conventions of music typography.
12. *Scribe Document Production System User Manual*. Unilogic, Pittsburgh, Pa., 1984. Documentation for the Scribe text-formatting system.
13. Smith, L.C. Editing and printing music by computer. *J. Music Theory 17*, 2 (1973). A discussion of Smith's music-printing system.

CR Categories and Subject Descriptors: D.3.2 [**Programming Languages**]: Language Classifications—*nonprocedural languages*; H.3.m [**Information Storage and Retrieval**]: Miscellaneous; I.7.2 [**Text Processing**]: Document Preparation—*format and notation, languages*; J.5 [**Arts and Humanities**]: *music*
General Terms: Design, Human Factors, Languages
Additional Key Words and Phrases: computer-readable musical scores, music printing

Received 10/85, accepted 1/86

Author's Present Address: John S. Gourlay, Dept. of Computer and Information Science, The Ohio State University, 2036 Neil Avenue Mall, Columbus, OH 43210; network addresses: gourlay@ohio-state.arpa (ARPANET), cbosgd!osu-eddie!gourlay (UUCP).