S. R. Holtzman
Department of Computer Science
University of Edinburgh
J.C.M.B., Kings Buildings
Mayfield Road
Edinburgh, EH9 3JZ
Scotland

Using Generative Grammars for Music Composition

Introduction

The use of computers in music has, to date, focused primarily on sound synthesis and signal processing. Organized research in computer music mostly has been concerned with the development of hardware and software models for the synthesis of sound and the description of instrumental and instrumental-like timbres using these models.[1] The application of computers in the process of music composition (that is, the generation of sound structures) has been for the most part neglected, though it is not a subject of lesser interest.

The concern of researchers using computers to investigate composition is to make the rules of music, or at least of a music language, explicit. A description of a composition process by explicit and formal rules may be used to program, automate, and simulate certain aspects of the process of music composition. Some composers, such as Iannis Xenakis (1971), have used computers in the process of composition as compositional aids, while others use the computer more consciously to study the way in which they compose, for example, G. M. Koenig (1970*a*; 1970*b*).

Perhaps the first investigation into programming compositional rules was done by Lejaren Hiller in the 1950s when he experimented with generating compositions by making random choices within specified bounds. The first "computer-composed" piece is his *Illiac Suite* (1958). Since the early 1960s Iannis Xenakis (1971) has formalized and programmed rules of his compositional method—notably his rules of *stochastic composition*. He has used Markov chains to generate strings of sound textures where each texture is in turn described by probability distributions. More recently, G. M. Koenig (1970*a*; 1970*b*) has programmed *serial* and other composition rules to generate compositions of impressive sophistication.

Though other work has doubtless been done, the brevity of the list of well-known significant work itself tells one something about the state of research in computer music composition.

In all of these cases, the composer has programmed compositional rules on a computer (in a formal and explicit manner). The computer, following the defined rules, simulates the composer's rules and generates compositions (or parts thereof). The adequacy of the description of the compositional process may be evaluated from the results.

A great barrier to composers and researchers attempting to formalize compositional rules—and rules of other musical activities—has been both the difficulty of making explicit these rules (i.e., what it is that people actually do when composing) and then expressing these rules to a computer.

A Generative Grammar Approach

In recent work a Generative Grammar Definition Language (GGDL) compiler, the GGDL compiler (Holtzman 1979*b*; 1980*a*; 1980*b*), has been implemented. In GGDL, the grammars of languages may be described by sets of formal and unambiguous rules. Special features have been included in GGDL in order to make it especially suitable for the description of music languages.

1. For example, models for sound synthesis include Fourier synthesis, frequency modulation (FM) synthesis (Chowning 1973), and VOSIM synthesis (Kaegi and Tempelaars 1978). Research in synthesizing instrumental timbres includes J. C. Risset's (1966; 1968) extensive work using additive and FM synthesis, J. Beauchamp's (1979) and D. Morrill's (1977) synthesis of trumpet timbres, and J. Grey's (1975) exploration of musical timbre.

Computer Music Journal, Vol. 5, No. 1, Spring, 1981,
0148-9267/81/050051-14 $04.00/0
© 1981 Massachusetts Institute of Technology.

A definition in the form of a GGDL program is sufficiently explicit that a machine can execute the instructions of the program to generate utterances in the described music language. A program complementary to the GGDL compiler, the GGDL-generator compiler-compiler, accepts the definitions of music languages in GGDL and may be used to generate utterances, that is, compositions, in the defined language.

The GGDL programs could be a valuable aid in both music research and computer composition. Without such a language, a researcher or composer is faced not only with the task of formalizing the rules of composition, but also of designing a means of expressing these rules to a computer. GGDL is an attempt to facilitate the programming of rules on a computer. By providing a framework and conceptual foundation for formalizing rules (i.e., formal grammars and language theory), GGDL should help make the formalization of the rules themselves easier.[2]

In this article, the GGDL language is used to demonstrate how the rules of a compositional language might be formalized for a computer. The examples are drawn from practical research done over the past four years—they reflect research into the formalization of compositional rules that has been implemented, tested, and that has produced some fruitful and interesting results.

The GGDL Generative Grammar Definition Language

A common way of formally representing the rules of a grammar is with *rewrite rules*. These rules take the form:

[X → Y]

where the arrow may be interpreted as an instruction to replace or rewrite the character string on the left-hand side (LHS) of the arrow with the character string to the right of the arrow.

The representation of language rules in GGDL is based on rewrite rules. Rewrite rules have been classified into four types (zero to three) by Noam Chomsky (1957). In the GGDL language, rewrite rules of type zero through three may be used. That is, an arbitrary number of elements (morphemes) may occur on both the LHS and right-hand side (RHS) of the rewrite arrow.[3] In a generative system, however, it will most likely be necessary to permit several alternative productions on the RHS and to allow the definition of complex selection functions to choose from among them.

In GGDL, using the period as an *alternative* marker, a rewrite rule permitting five different productions could be defined as follows.

[LHS → A . B . C . D . E]

This means LHS produces A or B or C or D or E. By default, selection is random. However, GGDL has a number of facilities that enable the use of more complex selection functions. For example, by inserting a shriek (exclamation point) just after the rewrite arrow, selection is serial—that is, no object is selected a second time until all others have been selected once. This is useful for musical purposes.

[LHS → ! A . B . C . D . E]

Alternatively, a production may be defined in terms of a finite-state transition matrix, where successive productions are dependent on previous productions (from the same LHS nonterminal).

```
[ LHS →
        ( A . 1 . 0 . 1 . 0 )
        ( B . 0 . 1 . 0 . 0 )
        ( C . 1 . 1 . 0 . 0 )
        ( D . 1 . 1 . 1 . 1 ) ]
```

2. In his article "Grammars as Representations for Music," C. Roads (1979) surveys work in music research where grammars have been applied. A theoretical foundation for the GGDL language is given in "Music as System" (Holtzman 1978*a*).

3. For a full discussion of grammars and rewrite rules, any standard text on linguistics should be consulted, for example, *Syntactic Structures* (Chomsky 1957) and *Introduction to Theoretical Linguistics* (Lyons 1968). A specification of the GGDL language may be found in my doctoral thesis (Holtzman 1980*c*). GGDL is also described in "A Generative Grammar Definition Language for Music" (Holtzman 1980*a*).

In such a transition matrix, each possible production from the nonterminal (LHS) is associated with a row of transition probabilities indicating the probability of the production being followed by the other possible strings that may be generated.

Should these system-provided selection mechanisms prove unsuitable for the description of a generation process, it is also possible to define *selection functions* in a high-level Algol-like language. Such selection functions return an integer value that is used as an index to select one of the RHS strings. *Meta-production* rules may also be defined in GGDL (Uzgalis 1977). Meta-production rules are used to indicate where a single generation by a rule is to be substituted for all occurrences of the LHS of the rule.

In the GGDL generative system the generation process is divided into three distinct stages. First, an abstract structure is generated representing relationships among musical objects. This process consists of the generation of utterances in a language where the language syntax is defined by rewrite rules and selection functions control the process of rewriting. Second, given a structure generated by such rules, the generation process undergoes a transformation process. *Transformational change markers* may be used to initiate musical transformations (such as inversion, transposition, and retrograde) on strings of objects. Finally, objects (terminals) are mapped (i.e., morphologically realized) from the abstract domain (as morphemes) to defined representations (as sounds). This third stage permits morphemes to be defined as any musical objects, as individual sound samples, and so on.

Applying a Grammar to Describing a Schoenberg Trio

The following is an example of a grammar description for music.

Compositional Rules

```
[ COMPOSITION → CANON ]
[ CANON →
  VOICE1,STRUCTURE,VOICE2,STRUCTURE ]
[ STRUCTURE → # 4 # TRANSFORMATION
  (GROUP) ]
[ TRANSFORMATION → ! – . @R . @I . @RI .
  @O6 . @R6 . @I6 . @RI6 ]
" GROUP → # 12 # ! OBJ1 . OBJ2 . OBJ3 . OBJ4 .
  OBJ5 . OBJ6 . OBJ7 . OBJ8 . OBJ9 . OBJ10 .
  OBJ11 . OBJ12 "
```

@ indicates a transformational change marker. Enclosing rules in double quotation marks rather than square brackets indicates that the rule is a meta-production rule; where a meta-production rule is used, one production of the LHS is substituted for all occurrences of the LHS string.

This set of rules will generate, among a large number of other compositions, the pitch structure of Schoenberg's Trio (Fig. 1) from the *Suite für Klavier Op. 25* (1925). An example of another structure generated from the rules is found in Fig. 2. This is in fact the first structure generated using the rules and the GGDL composition system.

The duration structure of these compositions similarly could be described by a set of rules. For example, in all but the third group in each structure, (1) the first six objects of the group are the same duration, the last (i.e., the sixth) of the subgroup being tripled, and (2) the rest of the objects in the group share the same duration—half the duration of the first six objects—with, again, the last object's duration tripled. This could be notated as follows.

```
[ DURATIONS OF A GROUP → # NUMBER OF
  OBJECTS IN GROUP # ⟨SELECTDURATION⟩
  DURATION 1 ( ♪ ) . DURATION 1 * 3 ( ♩. ) .
  DURATION 2 ( 𝅘𝅥𝅯 ) . DURATION 2 * 3 ( ♪. ) ]
```

SELECTDURATION is an integer function that returns a number indexing the RHS string to be selected. If duration 1 is an 8th note and duration 2 is a 16th note, the function to generate the rhythmic pattern of the first, third, and fourth occurrences of the series could be the following.

Fig. 1. An analysis of the Trio from Suite für Klavier, Op. 25 *(Schoenberg 1925). Each version of the series is identified as follows: O — original; I — inversion; R — retrograde; RI — retrograde inversion; and 6 — tranposed six semitones. The ordinal values of the members of the series have also been marked; note that in R_6 and RI_6 the seventh and eighth notes of the series are reversed.*

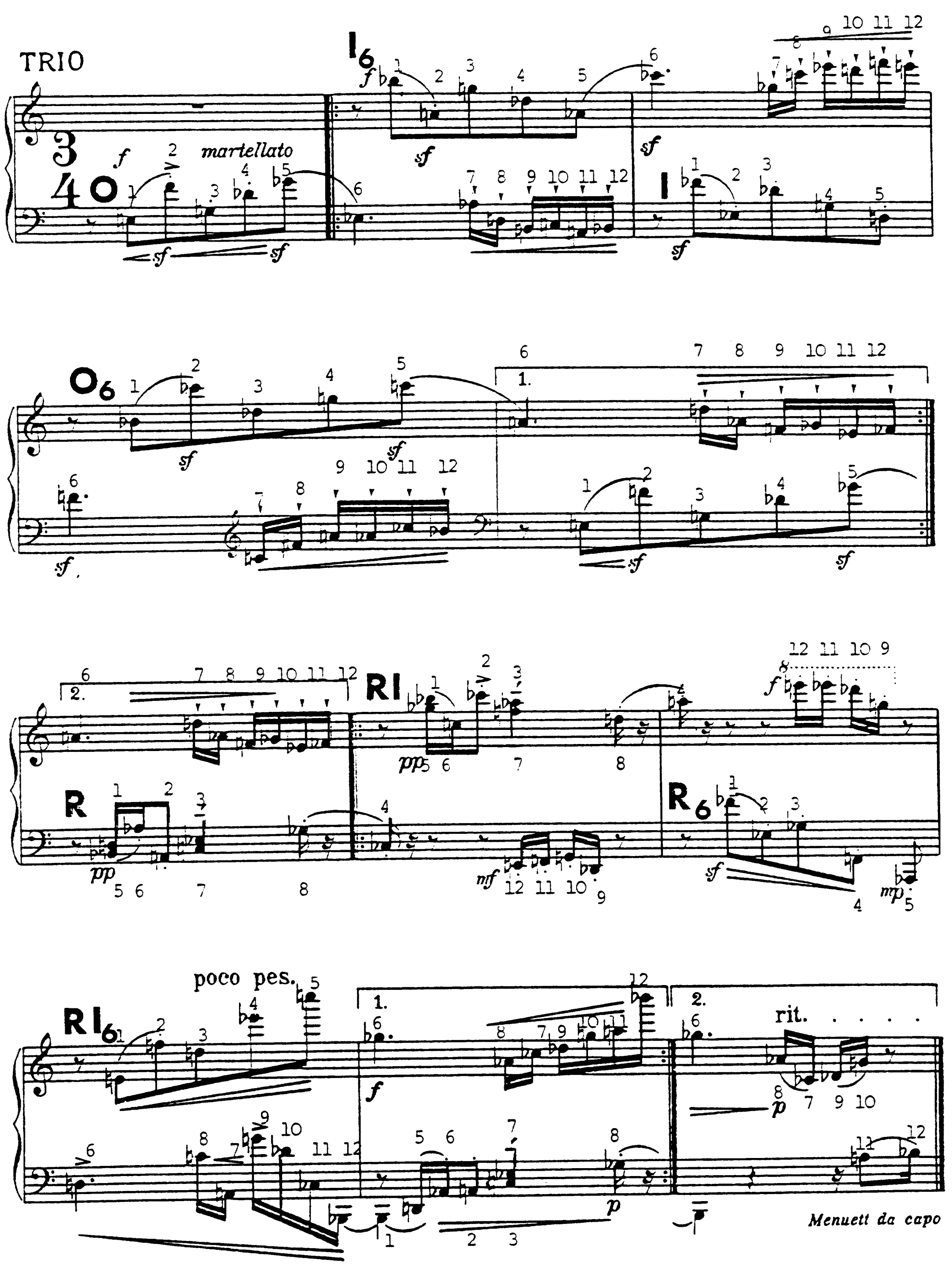
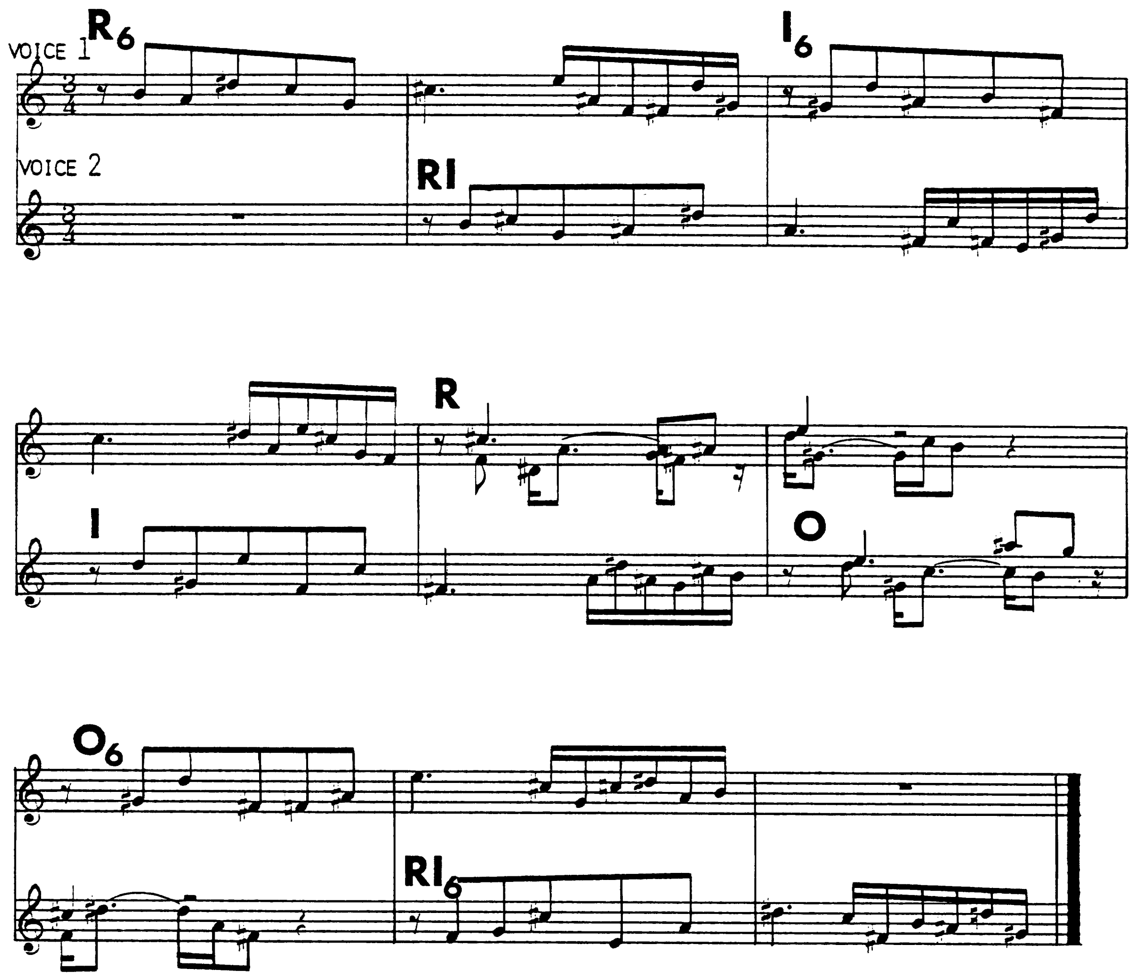


Fig. 2. A structure generated from a grammar that includes in the language it defines the Schoenberg Trio of Fig. 1: After Schoenberg.

Fig. 2a. A transcription of the data generated by the computer.

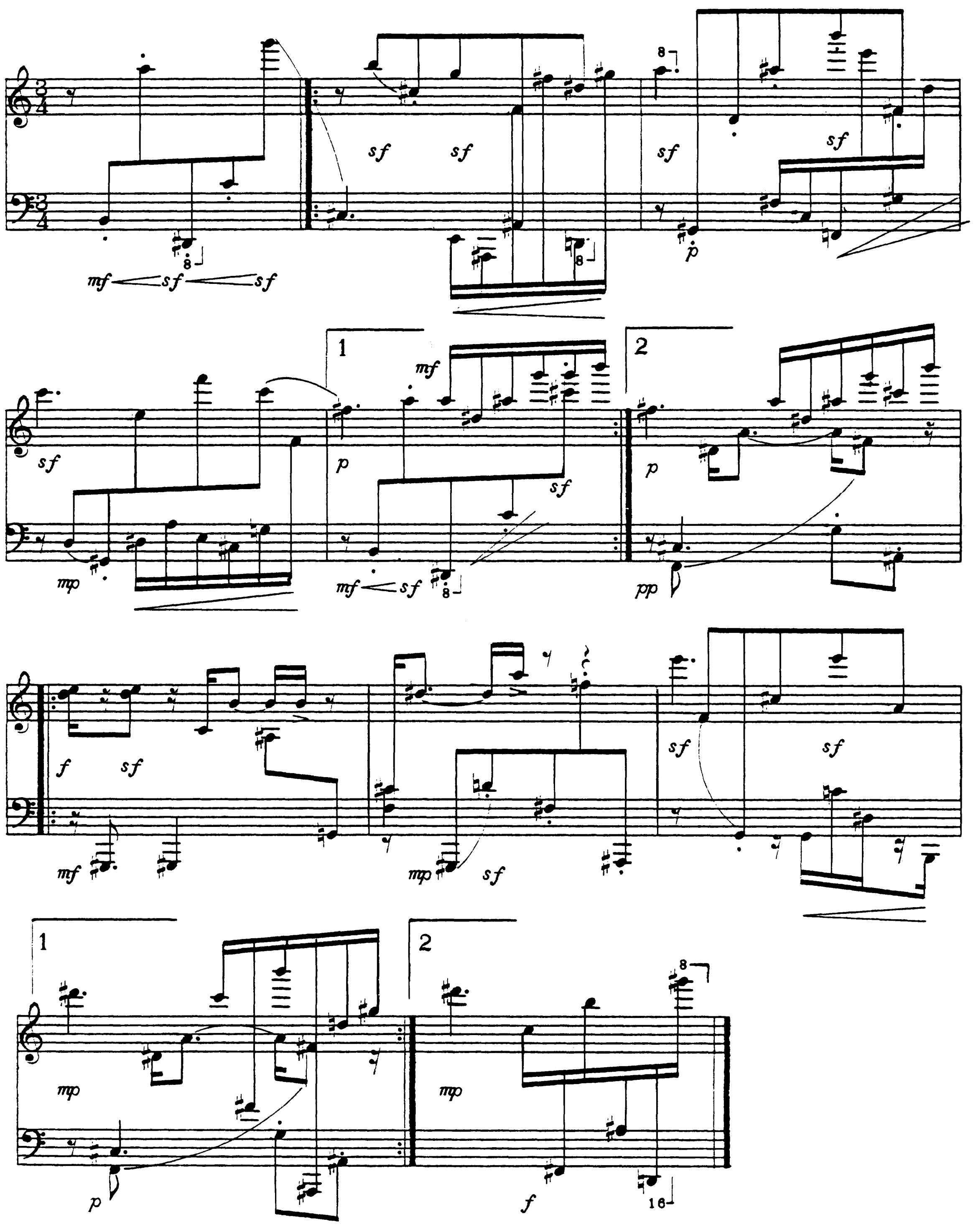


```
FUNCTION SELECTDURATION
  OBJCNT=OBJCNT+1  ;! a global variable counting
                    ! objects in a series
  IF OBJCNT ≤ 6 THEN
    IF OBJCNT = 6 THEN RESULT=2 ELSE RE-
      SULT=1 FINISH
  ELSE
    IF OBJCNT = NUMBER OF OBJECTS IN GROUP
      THEN RESULT=4
    ELSE
      RESULT=3
    FINISH
  END
```

A set of rules for generating the duration structure of the third group in each structure could be defined. Using a finite-state transition matrix with four duration values, the following meta-production rule (using a meta-production rule ensures that the generated string is the same for both parts of the canon) generates the durations of the third series of each part in the compositions in Figs. 1 and 2.

```
" DURATIONS FOR THIRD SERIES →
  ( DURATION 2 ( ♪ ), . 3 . 1 . 0 . 0 )
  ( DURATION 1 ( ♪ ), . 0 . 0 . 2 . 2 )
  ( DURATION 3 ( ♩ ), . 0 . 1 . 0 . 0 )
  ( DURATION 4 ( ♩. ), . 1 . 0 . 0 . 0 ) "
```

Fig. 2b. An alternative transcription of the data.



An Application in Original Composition: *After Artaud*

Just as one could define a set of rules for generating relationships between objects that are whole events, one can define sets of rules for generating descriptions of these events at a micro level. The GGDL grammar system manipulates abstract tokens. These tokens can be complete sections of a work, the notes or objects that make up a section, or the micro components of those notes or objects. The object of the compositional process (whether automatic or manual) is somewhat ambiguous—one can compose sound structures, sound events, or the individual sounds of a piece of music (Koenig 1978). In GGDL one may define an arbitrary compositional object. Composition rules may be used to manipulate notes and durations or to manipulate micro-sound objects.

Rather than describe sounds in terms of sample relationships, as in the experiments of Xenakis (1971) and Koenig's SSP program (Berg 1978; Banks 1979), one could use a parametric synthesis model for the generation of sounds. For the composition of *After Artaud*, the FM model (Chowning 1973) was used for describing sounds. Using FM, a number of parameters may be dynamically assigned values that control the generated sound. These are the frequencies of the carrier and modulating oscillators, the amplitude of the carrier wave, and the modulation index, which determines the depth of modulation. The dynamic values for each of these parameters may be represented by envelopes.

Using GGDL, one could define a set of rules for generating such envelopes. An envelope, for example, could be described as consisting of an attack, a steady-state, and a decay.

[ENVELOPE → ATTACK, STEADY-STATE, DECAY]

The components of the envelope could then be defined. The envelope could, for example, be represented by a set of points (*x*,*y*) in a Cartesian plane, representing the turning points of the envelope. Different qualities of attack could then be described in terms of the gradient between the starting point of the envelope, e.g. (0,0), and the point at which the steady-state of the envelope begins, such as in the following, for example.

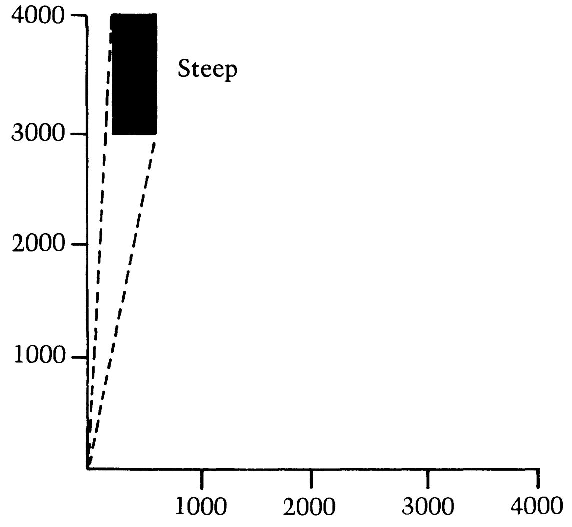


Fig. 3. Steep refers to a class of attacks that may be realized within a range of gradients.

[ATTACK → STEEP . MODERATE . SHALLOW]

Initializing the first envelope point to X=0 and Y=0, a steep attack could be described as X + a small increment and Y + a large increment

X = X + random(200,400)
Y = Y + random(2000,3000)

with an envelope window of, say, 4000 by 4000. Steep refers to a class of attacks that may be realized within a range of gradients (Fig. 3). Similar rules could be defined for generating different types of steady-states and decays.

For the composition of *After Artaud*, a grammar was defined for generating different types of envelopes, and then at a higher level rules were specified for combining different envelopes for the different parameters of an FM representation of a sound. Different types of envelopes were defined by their quality of attack, steady-state, and decay. For example, one type of envelope might have a steep attack, short steady-state, and slow decay, while another might have a shallow attack, long steady-state, and rapid decay. Different types of sounds are generated with different combinations of envelopes. For example, bell-like sounds have envelopes with very steep attacks and very gradual decays for both their modulation index and amplitude.

The *abstract-structure* of *After Artaud* was derived from an Artaud poem and was represented as a string of nonterminals, each representing a single sound. The nonterminals were rewritten as a string of terminals representing a set of envelopes, defined as (x,y) turning points, defining a sound that could be synthesized by means of the FM technique. For each unique sound object in the macro structure (i.e., nonterminals that occurred only once), the computer generated a distinctive set of FM characteristics—this was a unique set of carrier frequency, modulation-to-carrier-frequency ratio, and amplitude modulation (AM) and FM index envelopes. For sounds that were repeated the computer generated the same terminal representation of a description of the sound. During the mapping of the terminals, however, different values that would still have the same characteristics were generated. If a sound recurred in the macro structure (e.g., obj1 in obj1, obj2, obj3, obj1), each occurrence of the sound was realized differently. For example, if the modulation index of obj1 was described as steep attack, short steady-state, and shallow decay, the actual increment (i.e., steepness) of the envelope for each occurrence of obj1 was recalculated each time obj1 was mapped (i.e., realized) as an FM sound object. One could therefore get two modulation index envelopes with the same characteristics (perceptually identifiable as the same) yet slightly differing (Fig. 4). Every time a trumpet plays a C sharp, every time one pronounces the phoneme [ae], and every time the machine synthesizes obj1, slightly varying envelopes for parameters are generated though the object remains distinctive. It is still heard as C sharp, [ae], or obj1.

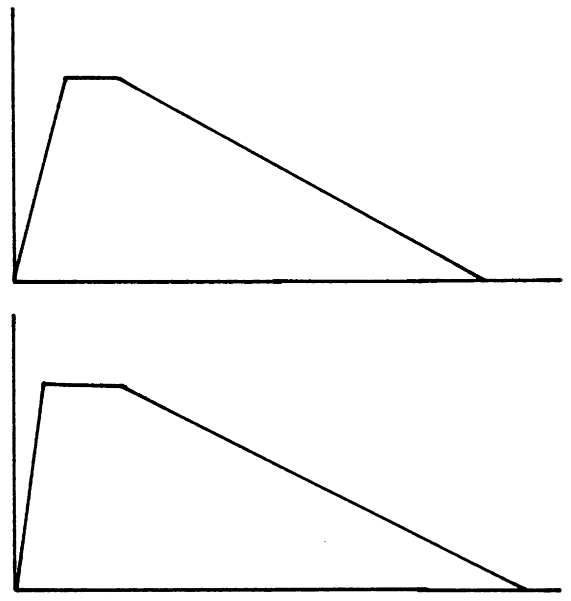


Fig. 4. Two envelopes that have the same characteristics yet differ slightly (e.g., steep attack, short steady-state, and shallow decay).

Incorporating Compositional Rules in a Program

The composition of music consists of more than just generating formal, abstract music structures. When writing a piece of music, a composer does much more than generate note and duration structures from a set of formal compositional rules. In the Schoenberg Trio example the set of rules generates only a note-duration structure. The Schoenberg composition, however, consists of much more than this. The octave distribution of the pitches has not been considered, nor have the dynamics or perhaps most important, the articulation of the work. Schoenberg's rule of serialism, that is, that no note (or whatever) should occur a second time until all others have occurred once, may be easily formalized and programmed. But it is a much greater problem to formalize other aspects of Schoenberg's style. For though he used serial rules to generate notes, rather than late-romantic harmony, Schoenberg's *Suite für Klavier* can be said, nonetheless, to be written in a Brahms-like piano idiom.

Most research in computer composition has been concerned with the generation of notes, durations, octave distribution, dynamics, and so on. There has been little attempt to try to incorporate in a set of compositional rules *knowledge* about what instruments and what sounds are used in the composition. A composer writing a piece of music for piano is likely to write a different piece than he or she would for violin: phrasing, octave distribution, dynamics, and the pitch structure itself are likely to be influenced by the medium by which the composition is to be realized. The following dialogue exemplifies the current concern with musically intelligent programs.

> C. ROADS: "One kind of artificial intelligence task is that of a program itself knowing what kind of sounds that it's actually dealing with and altering the program logic according to these sounds. Do you see this as a possibility?
> G. M. KOENIG: "Not only as a possibility, I think it is even a necessity . . . you need some kind of relationship between the musical language structure and the structure of the sounds produced" (Roads 1978*b*).

Further research using computer-assisted composition facilities such as the GGDL programs could investigate and formalize the rules of orchestration and of how the process of generating a music structure is related to the medium for the generated structure's performance.

Computer-Assisted Composition: *For Solo Harp*

A program was developed for the composition of a piece for harp, *For Solo Harp* (Holtzman 1977*b*). It was an attempt to design a composition program that could be said to have some musical intelligence. The task of generating a composition for an instrument is not only to formalize the rules of a music language but to integrate into the compositional rules a description of the instrument that is to perform the generated structure.

To write the composition for harp, it was necessary to constrain the generation of music structures in accordance with the limitations imposed by the harp itself. The most obvious practical problem facing any composer writing chromatic music for the harp is the availability of only seven notes in the octave at any one time and the associated problems of pedaling. But ideally, in addition to the practical details, a description of an instrument would consider the more subtle possibilities of the different methods of performing an instrument, and how they sound and affect the realization of the composition in musical rather than practical terms. That is, how the idiomatic capabilities of an instrument might be exploited to musical effect.

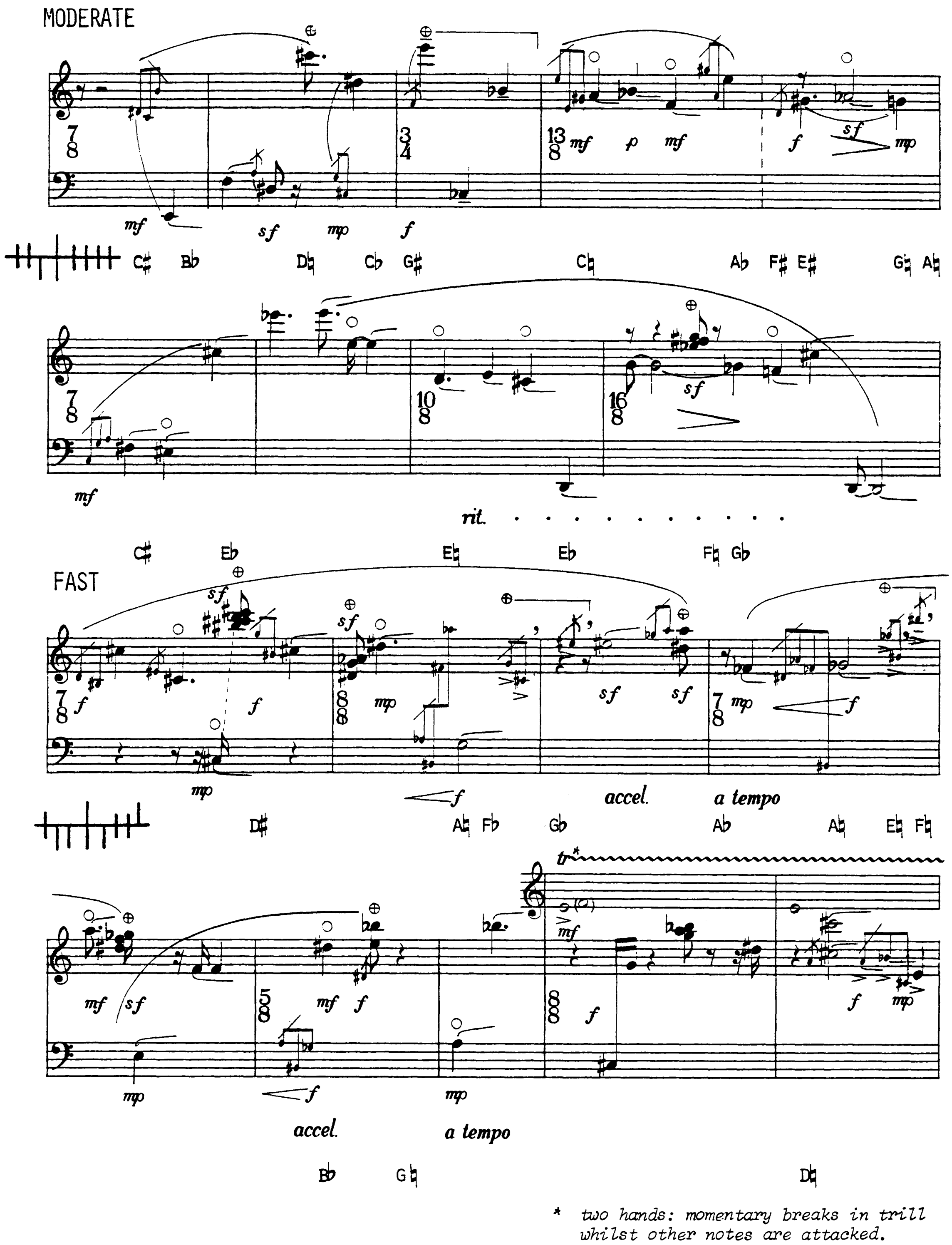
In the composition program used to write for harp a number of different ways of performing the harp were listed. The constraints associated with each manner of performance and the ways that one type of performed note could be related to another were also included. For each listed way of performing the harp, different variables could be given for octave distribution, dynamics, and possible pedal changes.[4]

When the computer generated a note of the composition, it would evaluate the context and determine what modes of performance would be suitable (i.e., sound good). Having selected, for example, a harmonic, it would determine what pitches could be performed and which octave would be suitable, which dynamic, and so on. It might consider, from a practical viewpoint, that a harmonic cannot be performed on the wound strings of a harp. From the standpoint of sound quality, it might assess that a harmonic will not sustain if played in the upper octaves, or that a full-bodied harmonic may be obtained by its performance in the mid-range of the harp. On the other hand, in performing chords, one can obtain a very brittle quality by playing a chord in the higher octaves of the harp. The program was designed to exploit these idiomatic qualities of the instrument to musical effect.

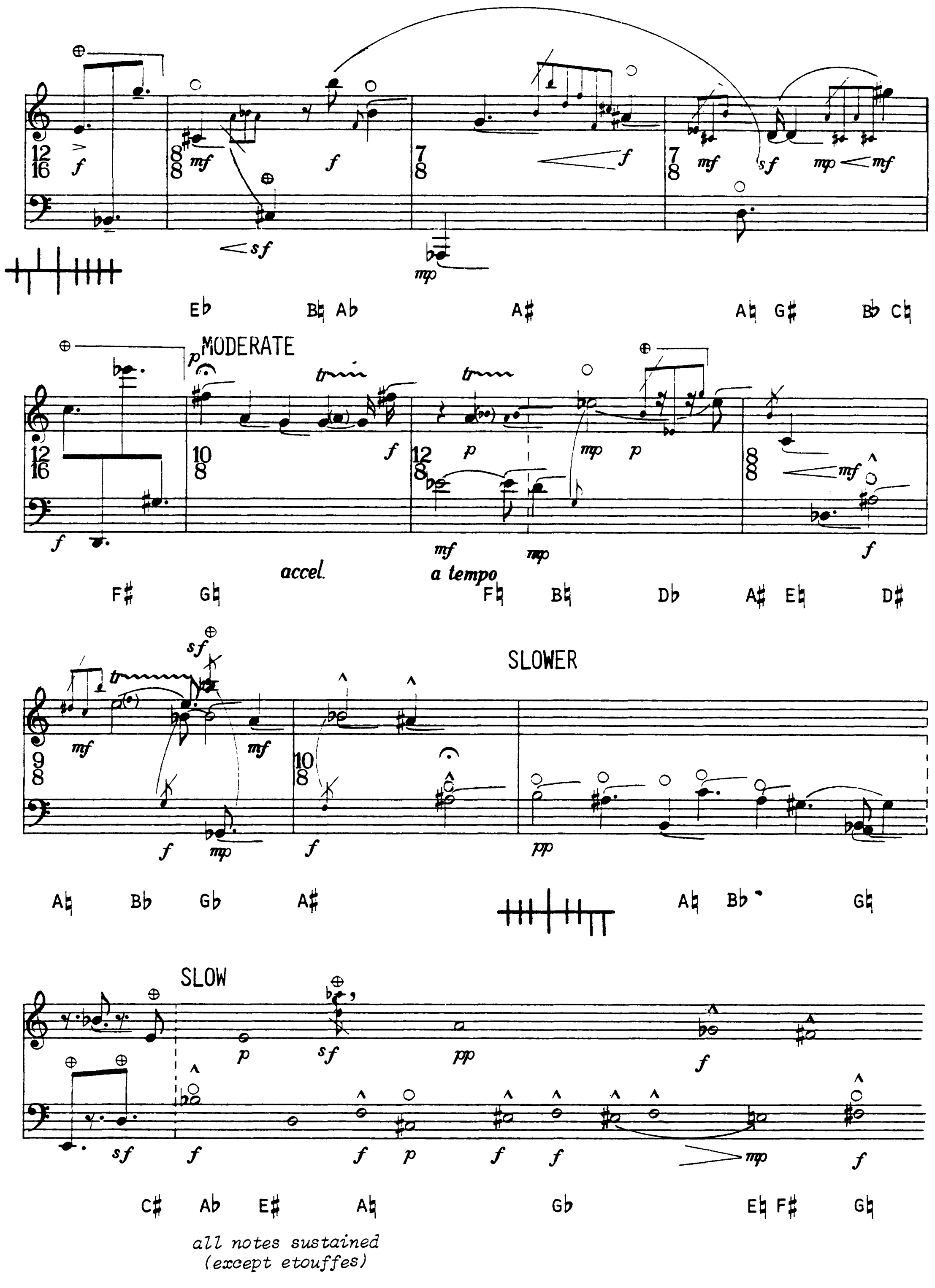
This is not necessarily the order in which a composer would make decisions without the computer. It is, however, one way of beginning to formalize the problems of composing for harp. The harp is a very difficult instrument for which to write, and in the program described the description of performing the harp was not completely successful. (As in any real as opposed to theoretical or ideal situation, the composition of the piece was constrained by the fact that it was commissioned for a specific instrument—in this case a difficult one for which to

4. It may be possible to change the available pitches to permit a wanted but not (at the time) available note to be performed, for example, if one is slowly performing harmonics or "ordinary" attacked notes. But the limitations increase if one is performing grace notes or chords, or if the tempo is fast; there may not be time to pedal-change.

Fig. 5. For Solo Harp *by S. R. Holtzman, composed with the aid of a computer.*

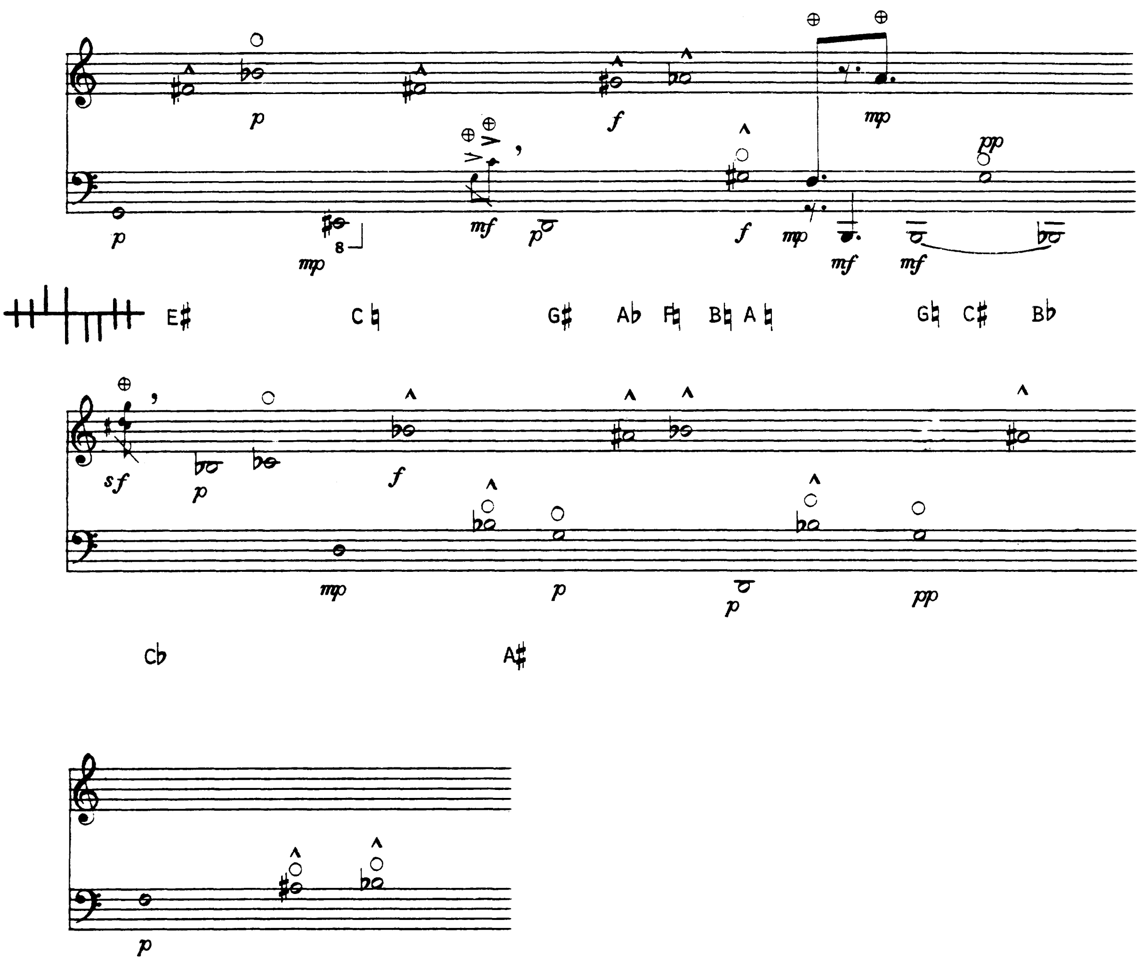


* *two hands: momentary breaks in trill whilst other notes are attacked.*



Eb Bâ®Ž Ab A#
Aâ®Ž G# Bb Câ®Ž
MODERATE
accel.
a tempo
F# Gâ®Ž Fâ®Ž Bâ®Ž Db A# Eâ®Ž D#
SLOWER
Aâ®Ž Bb Gb A#
Aâ®Ž Bb Gâ®Ž
SLOW
C# Ab E# Aâ®Ž Gb Eâ®Ž F# Gâ®Ž
all notes sustained
(except etouffes)

Fig. 5 (Cont'd)



write!) The program generated some unperformable music. Problems relating to the pedaling were the most obvious shortcomings of the formalization of harp technique. It was necessary to rearrange some of the music generated. Yet the computer did generate some effective and competently written music for harp (Fig. 5).

To have rearranged the material generated by the computer was not to invalidate the results of using the computer. The computer is seen as a tool and may be used as a composer wishes to achieve his or her compositional aims. In any case, any grammar that a composer may define is likely to generate more than one composition: a grammar defines a language in which there may be many utterances. Ultimately, a composer must choose which generated utterances to use, how to interpret the data generated by the machine, and so on. The composer may be seen as a selector.

For example, in Koenig's *Übung für Klavier* (1968), generated using a computer program, three variants of each of the 12 structures of the piece are given in the score; the performer must select one of each of the variants of the 12 structures for a performance. Koenig has been interested in systematically studying what he calls *form-potential.* Compositional rules define formal relationships that may be realized in music; these possible realizations are a form-potential. In terms of grammatical description, it is suggested that the form-potential of a set of rules could be likened to the language defined by a grammar.

With regard to the potential of a set of compositional rules, Xenakis has suggested that "A composer is able to [investigate] the general problems that the new musical form (i.e., that described by a set of rules) poses and to explore the nooks and crannies of this form" (Xenakis 1971).

Further Extensions

A possibility, given such research and fruitful results, would be to further aid composers in the process of composition by integrating into a computer-aided composition system a database listing constraints on instrumental performance and perhaps information on how such constraints need to be considered in the process of composition. Perhaps one could simply add to a set of composition rules a reference to the database's information on the instrument for which the composition is to be performed; the composition system could automatically ensure that a generated composition would be performable. Generated structures could be tested against a set of design rules in a way similar to that with which integrated-circuit computer-aided design systems check a circuit design against specifications of electrical and physical constraints.

There is, however, a considerable difference between circuit design and computer composition. In circuit design consistent rules result in a uniformity that permits a more efficient fabrication process to be implemented. Designs may also be checked for logical (i.e., functional) consistency. In music, efficiency is not a primary criterion for design or composition. Nor can the correctness of the logic of a musical language be objectively evaluated. In music there is no right or wrong. Some music just sounds better!

References

Banks, J. D., et al. 1979. "SSP—A Bi-Parametric Approach to Sound Synthesis." Sonological Report. Utrecht: Institute of Sonology.

Beauchamp, J. 1979. "Brass Tone Synthesis by Spectrum Evolution Matching with Non-linear Functions." *Computer Music Journal* 3(2): 35–43.

Berg, P. 1978. "A User's Manual for SSP." Unpublished manuscript. Utrecht: Institute of Sonology.

Chomsky, N. 1957. *Syntactic Structures*. The Hague: Mouton.

Chowning, J. 1973. "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation." *Journal of the Audio Engineering Society* 27(7): 526–534 and (1977) *Computer Music Journal* 1(2): 46–54.

Grey, J. 1975. "An Exploration of Musical Timbre." Report No. STAN-M-2. Stanford: Stanford University Department of Music.

Hiller, L. 1958. *Illiac Suite*. Composition for string quartet.

Holtzman, S. R. 1977*a*. "A Program for Key-Determination." *Interface* 6(2): 29–56.

Holtzman, S. R. 1977*b*. *For Solo Harp*. Unpublished manuscript (rev. 1980). Edinburgh.

Holtzman, S. R. 1978*a*. "Music as System." *Interface* 7(4): 173–187.

Holtzman, S. R. 1978*b*. *After Artaud*. A 4-track computer music composition. Utrecht: Institute of Sonology.

Holtzman, S. R. 1979*a*. "An Automated Digital Sound Synthesis Instrument." *Computer Music Journal* 3(2): 53–62.

Holtzman, S. R. 1979*b*. "GGDL Manual—Generative Grammar Definition Language." Department of Computer Science Report. Edinburgh: University of Edinburgh.

Holtzman, S. R. 1980*a*. "A Generative Grammar Definition Language for Music." *Interface* 9(2): 1–48.

Holtzman, S. R., ed. 1980*b*. "Grammars and Computer Composition." *Proceedings of Computer Music in Britain*. London: EMAS, pp. 95–110.

Holtzman, S. R. 1980*c*. "Generative Grammars and the Computer-Aided Composition of Music." Doctoral thesis, University of Edinburgh Department of Computer Science.

Kaegi, W., and Tempelaars, S. 1978. "VOSIM—A New Sound Synthesis System." *Journal of the Audio Engineering Society* 26(6): 418–425.

Koenig, G. M. 1968. *Übung für Klavier*. Unpublished manuscript. Utrecht.

Koenig, G. M. 1970*a*. "Project 1." Electronic Music Reports 2. Utrecht: Institute of Sonology.

Koenig, G. M. 1970*b*. "Project 2—A Program for Musical Composition." Electronic Music Reports 3. Utrecht: Institute of Sonology.

Koenig, G. M. 1978. "Compositional Processes." Paper presented at the UNESCO Computer Music Workshop, Aarhus, Denmark, 1978. Ottawa: UNESCO.

Longuet-Higgins, H. C., and Steedman, M. 1971. "On Interpreting Bach." In *Machine Intelligence* 6. Edinburgh: Edinburgh University Press.

Lyons, J. 1968. *Introduction to Theoretical Linguistics.* Cambridge, England: Cambridge University Press.

O'Connor, J. D. 1973. *Phonetics.* London: Penguin.

Morrill, D. 1977. "Trumpet Algorithms for Computer Composition." *Computer Music Journal* 1(1):46–52.

Risset, J. C. 1966. "Computer Study of Trumpet Tones." Murray Hill, New Jersey: Bell Laboratories.

Risset, J. C. 1968. "An Introductory Catalog of Computer Synthesized Sounds." Murray Hill, New Jersey: Bell Laboratories.

Roads, C. 1978*a*. "Composing Grammars." In *Proceeding of the 1977 International Computer Music Conference.* San Francisco: Computer Music Association.

Roads, C. 1978*b*. "An Interview with Gottfried Michael Koenig." *Computer Music Journal* 2(3):11–15.

Roads, C. 1979. "Grammars as Representations for Music." *Computer Music Journal* 3(1):48–55.

Schoenberg, A. 1925. *Suite für Klavier, Op. 25.* Vienna: Universal Edition.

Uzgalis, R., and Cleaveland, J. 1977. *Grammars for Programming Languages.* Amsterdam: Elsevier North-Holland.

Xenakis, I. 1971. *Formalized Music.* Bloomington: Indiana University Press.