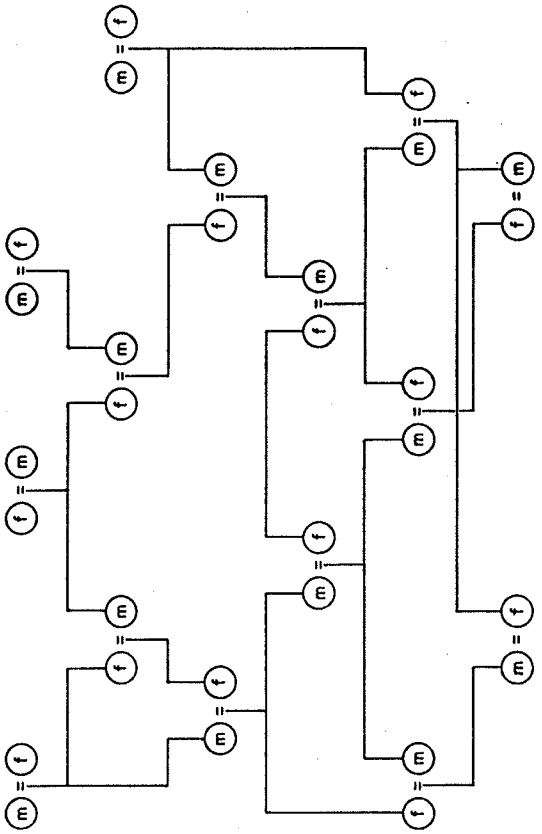
CHAPTER 11

MUSICAL DESIGN II: GRAMMARS

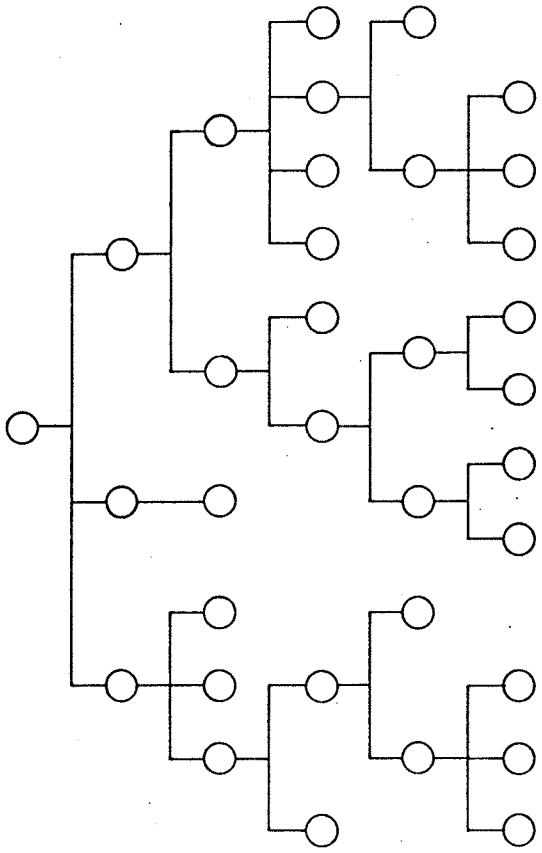
A grammar is a scheme of relationships which explains how the elements of a structure join together into the whole. The term itself derives from linguistics, which studies (among other things) the nested structure of languages: how syllables combine into words, words combine into phrases, phrases into clauses, clauses into sentences, sentences into paragraphs, paragraphs into chapters, and chapters into books (note 1). However, similar principles have proven very useful for explaining many non-linguistic structures, familiar instances including ~~religious~~ hierarchies, family trees, and musical designs.

Hierarchies exemplify a very important type of grammatic structure.

The word ~~hierarchy~~ properly denotes a religious organization ("hieros" is Greek for "sacred"), and most such organizations are characterized by the rule that each cleric has exactly one direct superior: in the Catholic Church, for example, a priest answers to his bishop, a bishop in turn answers to his cardinal, and a cardinal answers to the pope. In general, we shall designate any structure hierarchic if it obeys this



Family Tree



Hierarchy

Fig 11-1

rule. Hierarchies stand in contrast to structures whose units may have multiple superiors. This situation occurs in family trees, which are characterized by the (non-hierarchic) rule that each offspring has exactly two parents of mixed gender. Figure 11-1 illustrates hierarchic and non-hierarchic structures.
Networks = hierarchies + non-hierarchies.

Figure 11-1: Two grammatic structures - Diagram (a) illustrates a hierachic structure, while diagram (b) illustrates a geneological structure resembling a family tree. The letters M and F in diagram (b) indicate male and female offspring.

Perhaps the most well-known figure among linguists is Noam Chomsky. In his Syntactic Structures (1957), Chomsky emphasizes two basic types of relationship between grammatic units: phrase-structure relationships and transformations. Phrase-structure relationships associate general units with more specific ones. In typical applications, they elaborate upon (or distill) the content of an utterance while leaving the general structure unchanged; for example, given the general sentence "Dogs like people", we can substitute the specific cases "My German shepherds Soloman and Sheba" for "dogs", "positively adore" for "like", and "mailmen" for "people" to produce the specific case, "My German shepherds Soloman and Sheba positively

adore mailmen". Transformations change the structure of an utterance while leaving the content (in some sense) unchanged. A simple example is the transformation from active to passive voice: "Dogs like people", has the same content as "People are liked by dogs".

Chomsky further distinguishes between two types of phrase-structure relationship: context-free and context-sensitive. Context-free relationships depend solely upon information inherent in particular grammatic units and their subordinates while context-sensitive reltionships also incorporate environmental considerations. Chomsky limits the terms "context-free" and "context-sensitive" to hierarchic grammars; however, one can just as readily apply the distinction in non-hierarchic situations. For example, we know that genetic traits in children derive solely from their mothers and fathers, so can regard genetic relationships as "context-free". By contrast, character traits can also derive from aunts, uncles, teachers, other children, and so on; character relationships are therefore "context-sensitive". Another example is the Markov chain: a "0th" order Markov chain is context-free, while chains of increasingly higher orders reflect increasing sensitivity to the environment.

When a phrase-structure relationship enables us to deduce a specific concept such as "My German shepherds Soloman and Sheba"

from a more general concept such as "dogs", we call it a production. Synthesis, or derivation, denotes the process of taking a fairly general concept called an archetype (or axiom) and applying a series of productions in order to obtain a specific instance, or statement of this archetype. For example, it required a series of three productions to synthesize "My German shepherds Soloman and Sheba positively adore mailmen", from "Dogs like people". The inverse of a production is a reduction, which induces general concepts from specific ones. The inverse of synthesis is analysis, or parsing, which induces archetypes from statements.

11.1 APPLICATION: GRAMMAR OF A MUSICAL CODING SYSTEM

In order to illustrate a grammar which is directly relevent to both music and computers, we shall examine a musical coding system developed by Alan Ashton (1970). Ashton's direct pupose was to transcribe notated musical scores to be played by a computerized organ, but his code is also ideal for direct digital synthesis and it has been implemented in this capacity by the author (1979). Since the number of notes in a score is typically very large in either application, the code must necessarily be

extremely compact. For this reason, each letter in Ashton's code functions as an independent symbol. Table 11-1 explains the basic symbols, which have been modified slightly by the author in order to extend the capabilities of rhythmic brackets. The code also provides features for describing nuances of articulation as well as gradually evolving tempi and dynamics; these features have been omitted here for simplicity:

Table 11-1: Basic symbols of a musical transcription code (after Alan Ashton).

Figure 11-3 illustrates how this coding system might be used to transcribe measures 41-43 Igor Stravinsky's Variations: Aldous Huxley in Memoriam (1965). The music appears in Figure 11-2.

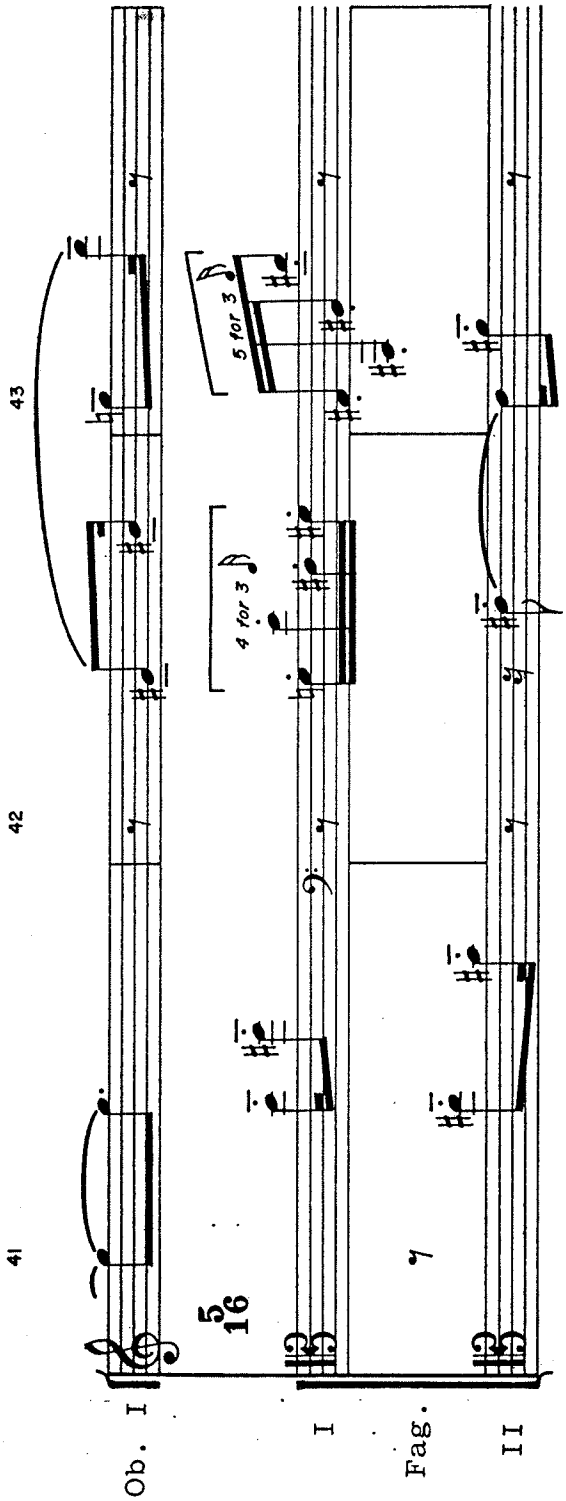
Figure 11-2: Igor Stravinsky, Variations: Aldous Huxley in Memoriam, measures 41-43.

Figure 11-3: Alphanumeric transcription of Stravinsky's Variations.

The grammatic units of the coding system, listed by decreasing level of generality, are:

Category	Type	Symbols
Digits		0 1 2 3 4 5 6 7 8 9
Letter names		A B C D E F G
Rest		R
Accidentals	Natural	%
	Flat	!
	Sharp	#
Chordal brackets		( )
Rhythmic neumes	Whole	W
	Half	H
	Quarter	Q
	Eighth	I
	Sixteenth	S
	Thirty-second	T
	Sixty-fourth	X
Other rhythmic symbols	Dot	.
	Tie	&
	Measure number	M<integer>
	Measure restart	;
	Barline	/
	Time signature	$<integer>-<integer>
	Rhythmic brackets	[ ]
Instrument		@<integer>
Layering brackets	Begin section	{
	Restart section	}

Table 11-1



Ex 11-2

1. compositions.
2. sections, layers,
3. measures,
4. parts,
5. tunes,
6. rhythmic strings,
7. rhythmic events,
8. chords,
9. integers, pitches (tied pitches), rests, durations,
10. characters (see Table 11-1),

Table 11-2 works its way upward through this list, defining each grammatic unit in relation to the more specific units. Notice that several definitions are recursive (chapter 10); the definition of a "tune", for example, admits nesting of rhythmic brackets to arbitrary depth, while the definitions of "sections" and "layers" allow large blocks of material to be spliced together and/or superimposed in ever-increasing hierarchies (note 2). The set of definitions taken as a whole describes the grammar of the coding system.

Figure 11-4 illustrates how Figure 11-3's alphanumeric transcription of measure 42 from Stravinsky's Variations is grammatically structured. Figure 11-4 may be read in either of two ways:

1. Reading Figure 11-4 from the top downward traces the behavior of an encoder as he derives details of his code from the gross notational structure of measure 42.

2. Reading Figure 11-4 from the bottom upward traces the behavior of the translating program as it parses the code into more general grammatic units.

It must be stressed that the nomenclature given in the above list is idiosyncratic to the coding system; a "tune" according to Figure 11-4's Definition 10 has little to do with a "tune" as conceived by a composer. Though the code strongly reflects the grammar of traditional musical notation, it gives few insights at all into true compositional relationships. Such relationships matter not at all to the translating program, which requires simply that the code be organized according to conventions which give each symbol meaning: "5%DH" means nothing while "HD%5" means a note (specifically, "half-note D natural five octaves plus a major second above 32' C") only because the translating program expects attributes of notes to occur in the following order: duration, letter name, accidental, register. In general, all encodings analyzable according to Table 11-2 will be "meaningful" to the translating program, while other encodings will invoke at least one error message.

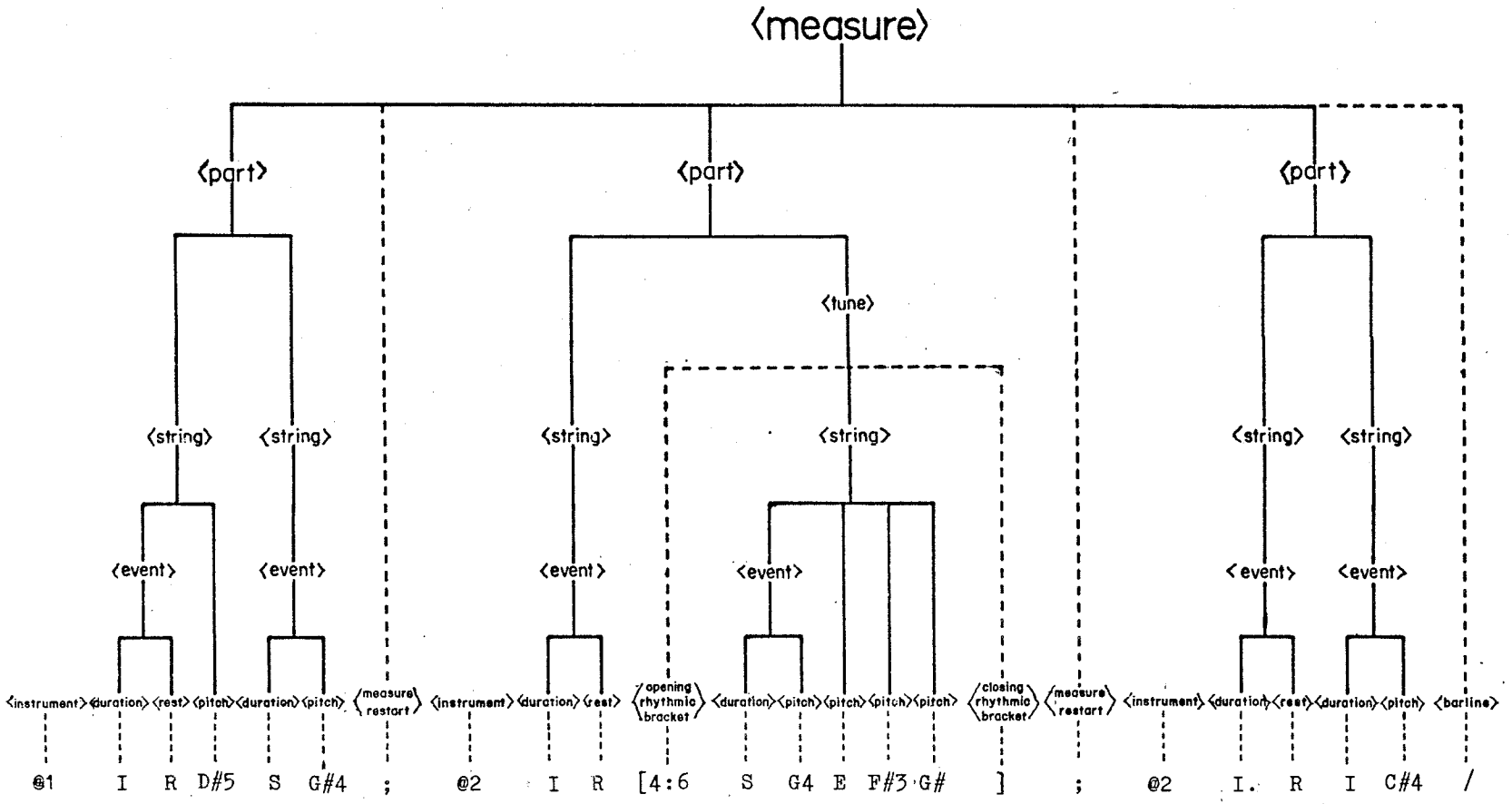

⟨measure⟩
⟨part⟩
⟨part⟩
⟨part⟩
⟨tune⟩
⟨string⟩
⟨string⟩
⟨string⟩
⟨string⟩
⟨string⟩
⟨string⟩
⟨event⟩
⟨event⟩
⟨event⟩
⟨event⟩
⟨event⟩
⟨event⟩
⟨instrument⟩ ⟨duration⟩ ⟨rest⟩ ⟨pitch⟩ ⟨duration⟩ ⟨pitch⟩ ⟨measure restart⟩ ⟨instrument⟩ ⟨duration⟩ ⟨rest⟩ ⟨opening rhythmic bracket⟩ ⟨duration⟩ ⟨pitch⟩ ⟨pitch⟩ ⟨pitch⟩ ⟨pitch⟩ ⟨closing rhythmic bracket⟩ ⟨measure restart⟩ ⟨instrument⟩ ⟨duration⟩ ⟨rest⟩ ⟨duration⟩ ⟨pitch⟩ ⟨barline⟩
@1 I R D#5 S G#4 ; @2 I R [4:6 S G4 E F#3 G#] ; @2 I. R I C#4 /

Fig 11-4

Figure 11-4: Grammatic structure of code for measure 42 of Stravinsky's Variations.

11.2 MUSICAL GRAMMARS

Musicians have been conscious of nested structures since at least the early nineteenth century. In his 1817 Theory of Composition, Gottfried Weber describes an approach to composing melodies in which a basic melodic line may be ornamented (using passing tones, neighboring tones, and so on) and the ornamental tones may themselves be ornamented up to arbitrary levels of complexity. The grammar associated with Weber's approach is functional in the sense that discrete notes serve directly as grammatic units on all levels of generality; that is, some notes are primary, some are secondary, others are tertiary, and so on. We have encountered non-musical instances of functional grammars in religious hierarchies and family trees, where each grammatic unit corresponds to an individual person.

The most widely celebrated functional approach to musical analysis is Heinrich Schenker's system of graphic ~~analysis~~ reduction (1935). Schenker generalizes Weber's approach to embrace harmony

as well as melody, and extends it to much more elaborate constructs than single-note ornaments. He conceives of musical structure as a sequence of lesser and greater goals, individually manifest as discrete chords. The more significant goals appear fairly infrequently and determine 'essential' direction, while other goals occur as detours along the way.

Standing in contrast to functional grammars are architectural (or "architectonic") grammars. Architectural grammars incorporate discrete elements only on their most specific levels; these elements combine and recombine into progressively larger aggregates as generality increases. While functional and architectural structures are closely related in the sense that every "functional" structure has an alternate "architectural" description, the differences are very prominent in practice. The grammatic units of natural languages such as English and of artificial coding systems such as Alan Ashton's are architectural, rather than functional; architectural approaches to explaining actual musical structure include the system devised by Alfred Lorenz (1924) to analyze the music of Richard Wagner, the procedures for rhythmic analysis devised by Leonard Meyer and Grosvenor Cooper (1960), and James Tenney's method of analysis based upon principles of Gestalt psychology (1964, 1980).

Though the majority of analytic approaches are strictly

hierarchic, recent work by Ray Jackendoff and Fred Lerdahl (1983) has applied non-hierarchic models to account for dualities of function: elided phrases, common chords in traditional modulations, and so on. Jackendoff and Lerdahl's approach is highly eclectic; they promote analyzing structures from several independent perspectives using functional approaches generalizing Schenker's and several architectural approaches, some generalizing Meyer and Cooper's, others related to Tenney's.

11.2.1 Grammatic Levels

There is a plethora of nomenclature used to distinguish grammatic levels. When designating levels by number, we conventionally refer to the archetype as level 1, to direct subordinates of the archetype as level 2, to subordinates of these subordinates as level 3, and so on. More general units traditionally reside at the "higher" levels of structure with the archetype occupying the "apex"; however, more recent practice has followed Chomsky by associating generality with "depth" and designating the most specific level(s) as the "surface". Schenker uses "background" to designate the level containing only the most significant notes, one or more "middlegrounds", as less

significant notes are included, and "foreground" for the most complete detail. In Tenney's nomenclature, a composition divides into "sections", "segments", "sequences", "clangs", and ultimately into "elements" (discrete sounds). The author prefers to retain the traditional association between generality and height while using geographic nomenclature to designate particular levels. It will therefore be the convention of this book to designate the most general level as "global", and to descend in generality through one or more median levels (for examples, "regional" and "parochial") to the "local" elements.

11.2.2 Self-Similarity

A structure may be designated self-similar when similar principles govern relationships at each grammatic level. Literal self-similarity (when 'the whole is reflected in the parts') sometimes projects an organic sense of unity, and this property has led authors such as Bolognesi (1983) to regard self similarity in itself as an aesthetic virtue. Tenney is less mystic, arguing that since (according to Gestalt psychology) the "factors" affecting how listeners perceive musical relationships are relatively independent of grammatic level, then analytic or

compositional procedures should be similarly independent.

11.2.3 Compositional Grammars

Where analysts invariably work inductively from the specific to the general, composers most often work deductively from general notions, elaborating these notions into a tangible creation. For our purposes it is sufficient to realize that the only information necessary to generate a grammatic structure is:

1. an archetype,

2. a set of one or more productions for deriving details from generalities, and

3. instructions for selecting productions whenever more than one production is applicable.

With this information, a program needs only to apply the productions at first to the archetype and then recursively to its own results until a fully detailed statement of the archetype has been derived. Hierarchies are the simplest grammatic

structures to program; to date, all compositions generated using grammatic automata have been hierarchic (including both of the demonstrations presented in this chapter).

11.2.3.1 James Tenney - James Tenney was the first composer to automate consciously hierarchic procedures using a computer. He implemented three-tiered hierarchies in the composing programs which produced his Phases for computer-generated tape (1963; described 1969) and his Music for Player Piano (1964). Both works divide at most general level into "sequences", each characterized by a set of randomly selected means and variances controlling such attributes of notes as duration, register, amplitude, and spectral content; in Phases, these random means and variances are themselves subject to the sinusoidal evolutions described earlier (heading 8.1.2). Each sequence divides into a number of "clangs"; the means and variances for the sequence served as parameters for random automata which generated new sets of means and variance for each clang. Clangs in turn divide into individual notes; the means and variances for the clang served as parameters for random automata which selected attributes for each note. Though Tenney's own programs were not recursive, such an implementation is now easily conceivable.

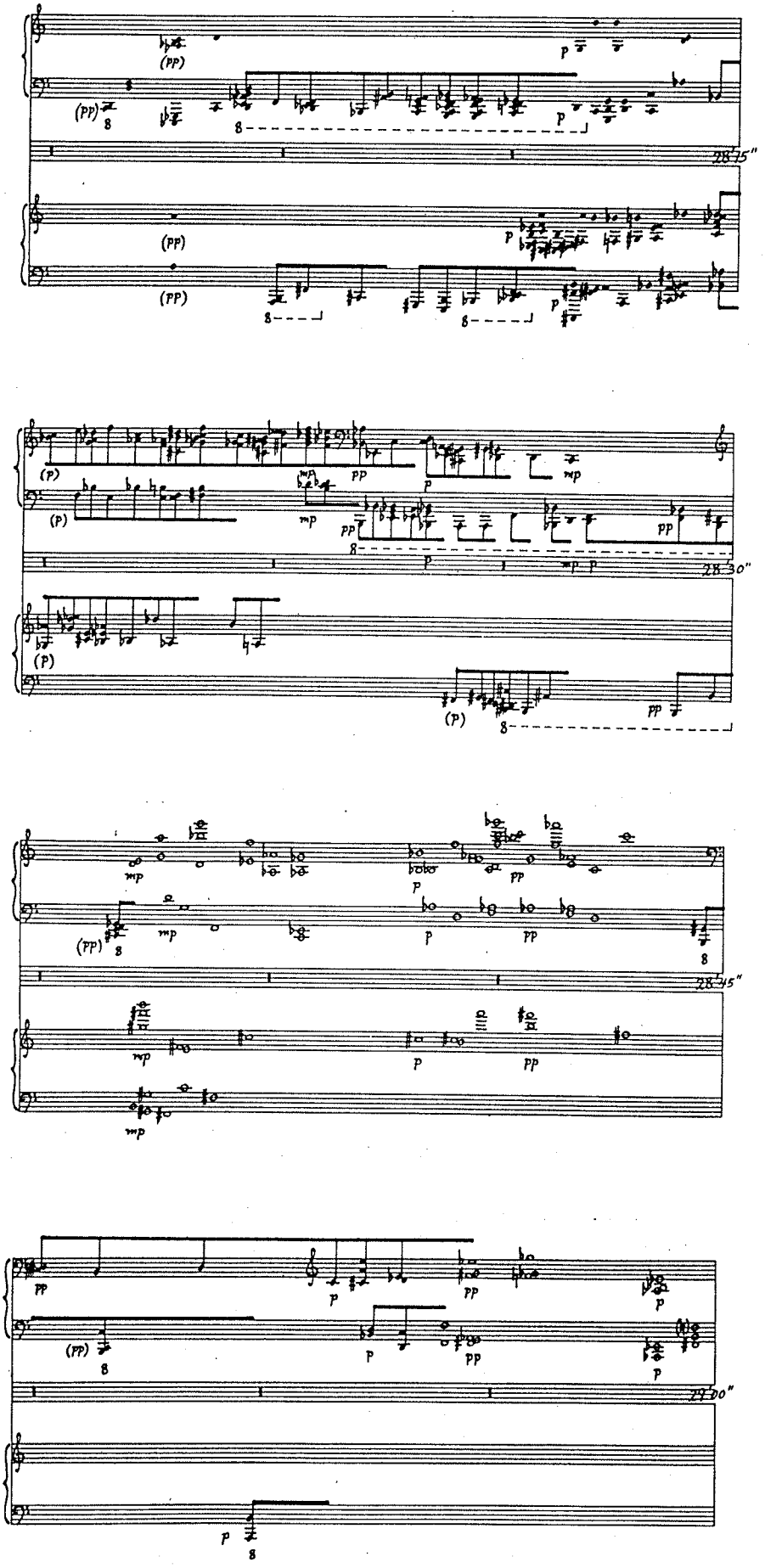


Fig 11-5

Figure 11-5: James Tenney, Bridge for two pianos, excerpt from 28'00" to 29'00". Copyright 1984 James Tenney.

With Bridge for two retuned pianos (1983/1984), Tenney inserts the intermediate level of "element" between notes and "clangs", so as to allow for chords, and extends his list of controlled attributes to embrace not only registers but also degrees. Bridge consists of three sections lasting approximately 8, 13, and 21 minutes, and Tenney's hierarchic "world" only becomes fully realized in the final section. The initial section projects a wholly random environment expressing the "world" of John Cage, while the middle section effects a bridge from one world to the other.

11.2.3.2 Formal ~~Approaches~~ Implementations - Curtis Roads and, later, Steven Holtzman have implemented fully recursive utilities for generating musical compositions from linguistic models. Road's (1978) utility enables a user to describe a composition by specifying an archetype along with a set of productions serving to deduce a specific statement of this archetype. All grammatic units are represented as abstract "tokens" whose functions are

defined by the way they are used. There are two kinds of token, "non-terminal" and "terminal", which Roads symbolizes using upper-case and lower-case letters, respectively. Productions take the form:

<non-terminal token> --> <string>

where string is any string of terminal and/or non-terminal tokens. Such productions may be interpreted as follows: rewrite each occurance of non-terminal token as string . Production are applied recursively until all non-terminals have been removed. For example, suppose a user specifies the archetype "ADA" and the non-recursive productions:

```
A  -->  B C B
B  -->  a b a
C  -->  c
D  -->  d
```

Given these productions, Road's utility would apply the sequence of rewrites depicted in Figure 11-6. Roads provides further capabilities for defining productions with several possible outcomes and for defining recursive productions, that is, productions which include the same non-terminal token on both sides of the arrow.

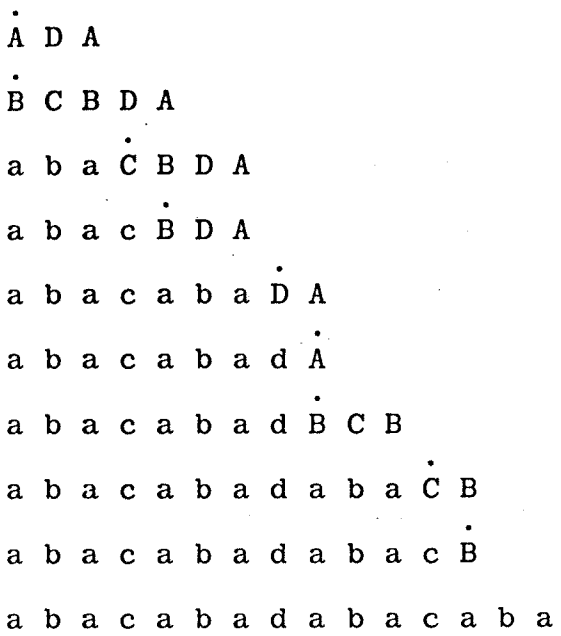

A D A
B C B D A
a b a C B D A
a b a c B D A
a b a c a b a D A
a b a c a b a d A
a b a c a b a d B C B
a b a c a b a d a b a C B
a b a c a b a d a b a c B
a b a c a b a d a b a c a b a

Fig 11-6

Figure 11-6: Generating of a statement from an archetype by non-recursive productions - Capital letters indicate "non-terminal" tokens; lower-case letters indicate "terminal" tokens. The dot above each line of text indicates which non-terminal character is next up for processing.

Up to this point in the process, the various tokens specified to Road's utility could just as easily represent positions on a corporate organization chart as musical functions. Statements achieve concrete musical meaning through a "mapping" between the set of terminal tokens and a set of acoustic "objects". The nature of these objects is defined by the user: they may be discrete pitches, recorded sounds, statistical "clouds", and so forth. In the simplest applications, each terminal token will correspond to a single, distinct acoustic object. However, the potential for less rigorous mappings gives rise to two intriguing notions defined by Roads as "musical synonyms" and "musical homonyms". Musical synonyms arise when one terminal token maps into two or more objects; musical homonyms arise when two or more terminal tokens map into the same object.

Steven Holtzman's "generative grammar definitional language for music" (1980) directly adopts Roads's approach to treating

musical units as abstract tokens. Where Road's utility is limited to context-free productions, however, Holtzman expands Road's capabilities to include context-sensitivity and weighted random selection. These features enable a user to describe conditionally random processes such as Markov chains, which require recursive, context-sensitive productions such as the following:

```
aX  -->  20% aaX  or  33% abX  or  12% acX  or  5% e
bX  -->  12% baX  or  20% bbX  or  33% bcX  or  5% e
cX  -->  33% caX  or  12% cbX  or  20% ccX  or  5% e
```

Figure 11-7 illustrates how a possible realization might be generated by the above set of productions from the archetype "aX". Notice that selecting an option containing the non-terminal "X", causes the process to automatically reinitiate a new production. Since each production provides a 5% chance of selecting the terminal option "e", the length of chains generated by the process follows a geometric distribution (heading 4.2,2,3) with an average of 20 transitions per chain.

Figure 11-7: Generating a Markov chain from an archetype by recursive productions with weighted random selection.

a $\dot{X}$

a c $\dot{X}$

a c b $\dot{X}$

a c b a $\dot{X}$

a c b a b $\dot{X}$

a c b a b a $\dot{X}$

a c b a b a b $\dot{X}$

a c b a b a b a $\dot{X}$

a c b a b a b a c $\dot{X}$

a c b a b a b a c b $\dot{X}$

a c b a b a b a c b c $\dot{X}$

a c b a b a b a c b c c $\dot{X}$

a c b a b a b a c b c c a $\dot{X}$

a c b a b a b a c b c c a b $\dot{X}$

a c b a b a b a c b c c a b c $\dot{X}$

a c b a b a b a c b c c a b c a $\dot{X}$

a c b a b a b a c b c c a b c a e

Fig 11-7

The generality of Holtzman's utility is evident in the fact that, given enough tokens, it is able to generate strings of arbitrary complexity which conform wholly to the rules of Ashton's musical coding system. Holtzman's utility also provides rudiamentary musical transformations -- inversion, retrograde, and transposition -- along with a facility for merging two motives into a composite.

11.2.3.3 ~~Contextual Approaches~~ Direct Implementations - In treating grammatic units exclusively as abstract tokens, Roads and Holtzman attempt to implement utilities which operate independently of a composer's particular stylistic prejudices. Theoretically, any structure with a finite number of terminals may be described abstractly; in practice, the productions may soon become prohibitively complex. A more expedient approach used by Kevin Jones and independently by the author is to develop specific programs which directly implement descriptive knowledge concerning each grammatic unit. This knowledge may then be manipulated numerically by the program.

Jones's method (1981) involves carving chunks of musical space into progressively smaller pieces until the program finally obtains descriptions of individual notes. Grammatic units are

described by pairs of boundaries in each musical dimension: starting and ending times t1 and t2, lowest and highest pitches p1 and p2, and softest and loudest dynamics d1 and d2.

$$\langle \text{unit} \rangle = \langle t1,t2;\ p1,p2;\ d1,d2 \rangle$$

Productions divide each unit into one or more sub-units by specifying new boundaries for each sub-unit, as in the following:

$$\langle t1,t2;\ p1,p2;\ d1,d2 \rangle \rightarrow \langle t1,t;\ p1,p;\ d1,d \rangle \text{ and } \langle t,t2;\ p,p2;\ d,d2 \rangle$$

where $t = (t1+t2)/2$
$p = (p1+p2)/2$
$d = (d1+d2)/2$

The author's composition Crystals for 16 strings (1980; described 1982) derives directly from Tenney's Gestalt theories, though it resembles Jones in its implementation. My procedures incorporate the use of feedback from previous levels to help insure that the products conform to certain initially prescribed characteristics. For example, if in a given section of the work it had been prescribed that only two simultaneous parts should be present, the program would check to see if a branch into two simultaneous parts had occured at a higher level. If so, then the program would not branch again. If not, then the program would branch with a probability 1/n, where n denotes the

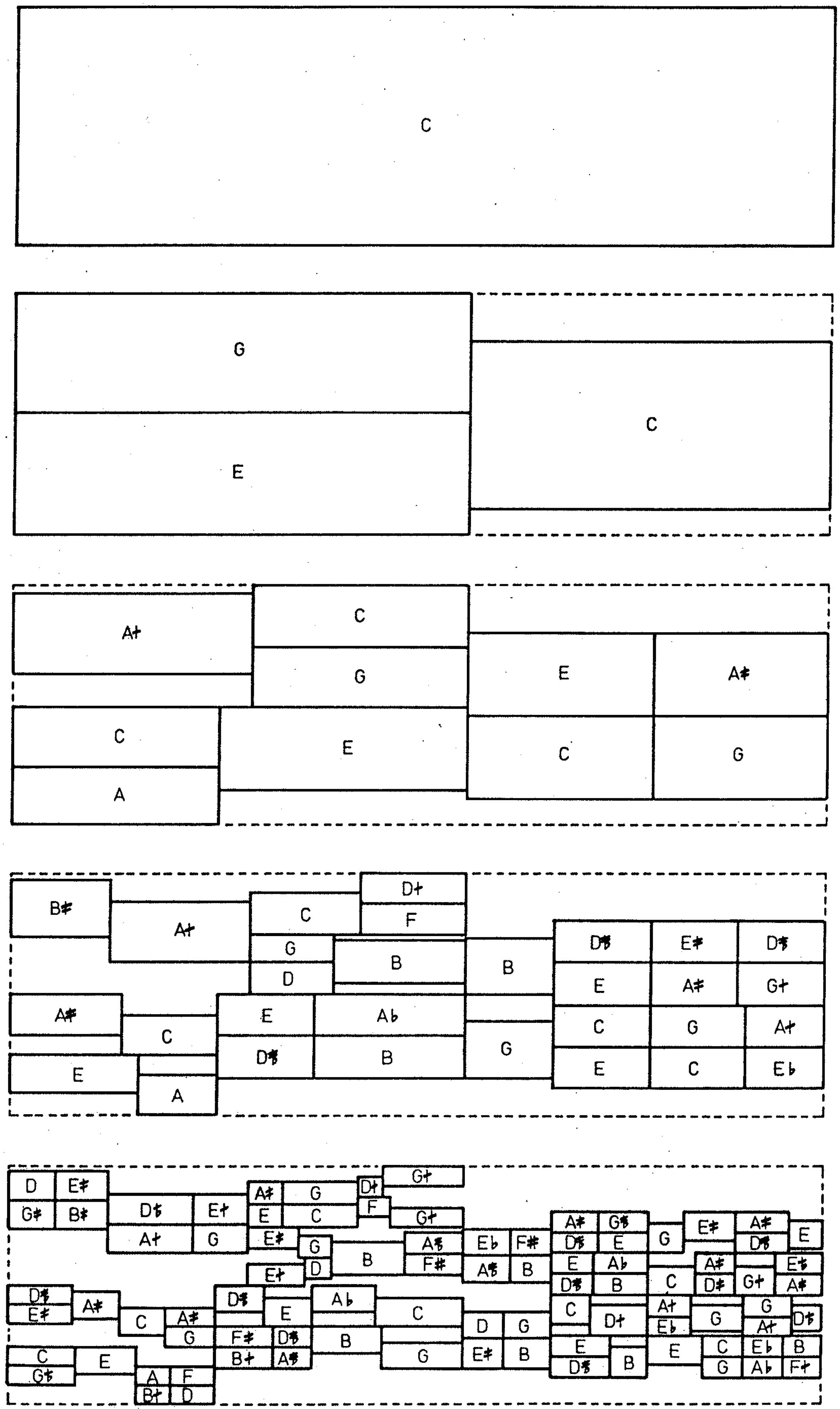

C
G
E
C
A+
C
G
E
A♯
C
A
E
C
G
B♯
A+
C
D+
F
G
D
B
B
D♯
E♯
D♯
E
A♯
G+
A♯
C
E
A♭
C
G
A+
E
A
D♯
B
G
E
C
E♭
D
E♯
G♯
B♯
D♯
E+
A+
G
A♯
G
E
C
D+
F
G+
E♯
G
D
B
A♯
F♯
E♭
F♯
A♯
B
A♯
G♯
D♯
E
E
A♭
D♯
B
G
E♯
A♯
D♯
E
A♯
D♯
G+
E♯
A♯
D♯
E♯
A♯
C
A♯
G
F♯
B+
D♯
A♯
E
A♭
B
C
G
D
E♯
G
B
C
D+
A+
E♭
G
G
A+
D♯
C
E
G♯
C
A
B+
F
D
E
D♯
B
E
C
G
E♭
A♭
B
F+

Fig 11-8

number remaining opportunities to branch (including the current level). Crystals also incorporates sensitivity to both degree and register. Figure 11-8 illustrates the process involved in generating one (hypothetical) section of this work.

Figure 11-8: Generative process in Charles Ames's Crystals (1980) - Reprinted from "Crystals: Recursive structures in automated composition", Computer Music Journal, volume 6, number 3 (1982). Copyright Charles Ames 1982.

11.3 DEMONSTRATION 8: A FUNCTIONAL GRAMMAR

Demonstrations 8 and 9 illustrate two ways of programming computers to generate complex musical statements given an archetype and a set of fairly simple productions. Demonstration 8 consists of three consective versions (variations) of a basic tune -- or in linguistic jargon, three consecutive statements of an archetype. Each successive variation ornaments the tune by increasing degrees of embellishment. The grammar of Demonstration 8 is "functional" in the sense that the notes of the basic tune directly constitute grammatic units at the highest

structural level, while all other notes may be regarded as "ornamental" with varying levels of significance. The productions used to embellish both the basic tune and the higher-level ornaments are drawn from traditional formulae: seconds are embellished by either "escaping tones" or "reaching tones"; thirds by changing-note groups ("cambiatas").

11.3.1 Compositional Directives

Figure 11-9: Basic tune of Demonstration 8.

Figure 11-10: Productions in Demonstration 8.

Figure 11-11: Transcription of Demonstration 8.

The basic tune is illustrated in Figure 11-9, while the ornaments appear in Figure 11-10. Notice that the basic tune and each of the ornaments consists solely of seconds and thirds. It is this constant property which enables the ornamenting process to be applied recursively up to five levels (counting the archetype as level 1): not only does the program elaborate upon the tune itself, but also upon the ornaments, upon the ornaments

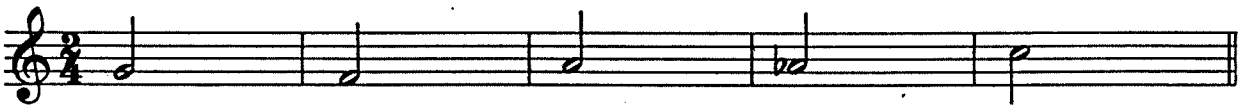


Fig 11-9

11-10

Demonstration 8

Clarinet

Charles AMES

STRICTLY ♩ = 80

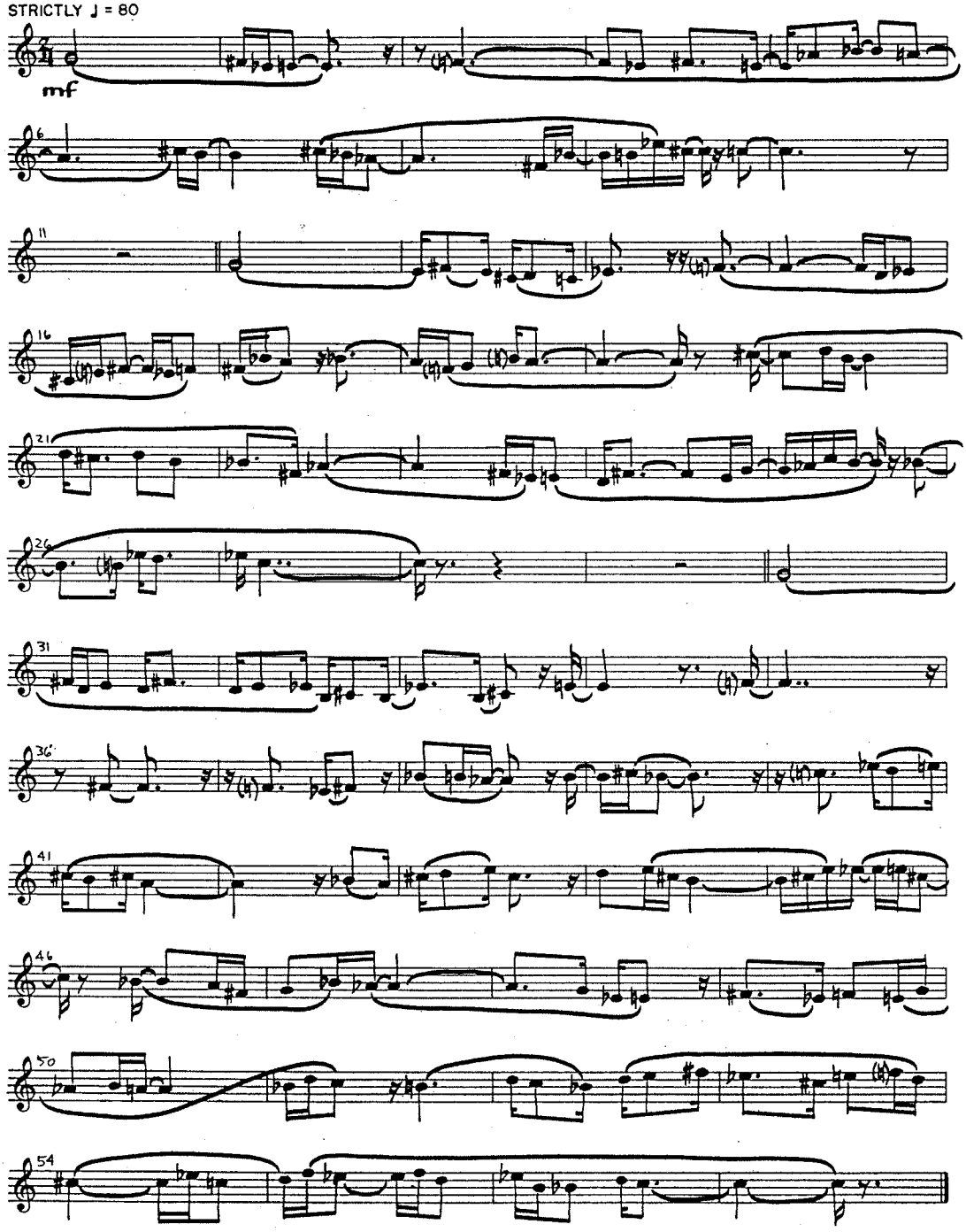


Fig 11-11

on the ornaments, and so on. Notes from the basic tune receive durations of 8 sixteenths; ornamental notes receive durations of 5, 3, 2, and 1 sixteenth, depending on their recursive levels.

11.3.2 Implementation

-- Programming example 11-1: program DEMO8 (3 pages) --

The trick to implementing a hierarchic composing process is to set up a stack (heading 10.2.2), or a collection of parallel stacks, so that all of the attributes characterizing a grammatic unit at an arbitrary level of recursion can be accessed using a single index. The composing program then resolves into a main loop which handles the level of recursion along with subsidiary procedures for deducing lower-level units from higher-level ones.

11.3.2.1 Main Program - Program DEMO8 proper keeps track of Demonstration 8's three versions and inserts a rest between each one. It also passes a value from array EMBVER to the first location of array EMBLEV; this value controls the number of nested ornaments.

```
      program DEMO8
C
C     Demonstration of functional grammar
C
      parameter (MPCH=5,MDEG=12,MLEV=5,MLEV1=6,MVER=3)
      real     EMBLEV(MLEV),CUMDEG(MDEG),DURLEV(MLEV1),RSTLEV(MLEV),
     :         EMBVER(MVER)
      integer TUNE(MPCH),IDXLEV(MLEV),CNTLEV(MLEV),PCHLEV(MPCH,MLEV)
      common  LEVEL,LEVEL1,IPCH1,IPCH2,
     :        TUNE,IDXLEV,CNTLEV,PCHLEV,EMBLEV,CUMDEG,DURLEV,RSTLEV
      data EMBVER/1.5, 3.0, 4.5/
C
      open (2,file='DEMO8.DAT',status='NEW')
      ITIME = 0
      IVER = 1
      do
C       Compose version
        EMBLEV(1) = EMBVER(IVER)
        call COMPOS(ITIME)
C       Increment version count
        IVER = IVER + 1
C       Test for end of composition
        if (IVER.gt.MVER) exit
C       Write pause between versions
        call WNOTE(ITIME,(16.-float(MOD(ITIME,8))),0)
      repeat
      close (2)
      stop
      end

      subroutine COMPOS(ITIME)
      parameter (MPCH=5,MDEG=12,MLEV=5,MLEV1=6)
      real     EMBLEV(MLEV),CUMDEG(MDEG),DURLEV(MLEV1),RSTLEV(MLEV)
      integer TUNE(MPCH),IDXLEV(MLEV),CNTLEV(MLEV),PCHLEV(MPCH,MLEV)
      common  LEVEL,LEVEL1,IPCH1,IPCH2,
     :        TUNE,IDXLEV,CNTLEV,PCHLEV,EMBLEV,CUMDEG,DURLEV,RSTLEV
C
C     Initialization
C
      LEVEL = 1
      IDXLEV(LEVEL) = 0
      CNTLEV(LEVEL) = MPCH
      do (I=1,MPCH)
        IPCH = TUNE(I)
        PCHLEV(I,LEVEL) = IPCH
        IDEG = MOD(IPCH,12) + 1
        CUMDEG(IDEG) = CUMDEG(IDEG) + DURLEV(LEVEL)
      repeat
      call WNOTE(ITIME,DURLEV(LEVEL),PCHLEV(1,1))
C
C     Main composing loop
C
      do
C       Increment index
        IDX = IDXLEV(LEVEL) + 1
        if (IDX.lt.CNTLEV(LEVEL)) then
          IDXLEV(LEVEL) = IDX
C         Determine starting and goal pitches
          IPCH1 = PCHLEV(IDX,LEVEL)
          IPCH2 = PCHLEV(IDX+1,LEVEL)
          LEVEL1 = LEVEL + 1
          if (IPCH1.eq.0) then
C           Cannot ornament a rest
            call WNOTE(ITIME,DURLEV(LEVEL1),IPCH1)
          else
            if (IDX.gt.1) call WNOTE(ITIME,DURLEV(LEVEL),IPCH1)
            if (EMBLEV(LEVEL).gt.0.0 .and. IPCH2.ne.0) then
              call DEDUCE
C             Advance to next level
              LEVEL = LEVEL1
              IDXLEV(LEVEL) = 0
            end if
          end if
        else
C         Backtrack to preceding level
          if (LEVEL.eq.1) exit
          LEVEL = LEVEL - 1
        end if
      repeat
      call WNOTE(ITIME,DURLEV(1),PCHLEV(MPCH,1))
      return
      end
```

Ex 11-1 (page 1 of 3)

```
      subroutine DEDUCE
      parameter (MPCH=5,MDEG=12,MLEV=5,MLEV1=6)
      real    EMBLEV(MLEV),CUMDEG(MDEG),DURLEV(MLEV1),RSTLEV(MLEV)
      integer TUNE(MPCH),IDXLEV(MLEV),CNTLEV(MLEV),PCHLEV(MPCH,MLEV)
      integer PTRORN(9),INCORN(9),NUMORN(9),ORNMNT(60),SCHED(6)
      logical SUCCES
      common  LEVEL,LEVEL1,IPCH1,IPCH2,
     :        TUNE,IDXLEV,CNTLEV,PCHLEV,EMBLEV,CUMDEG,DURLEV,RSTLEV
      data PTRORN/49,25, 9, 3, 0, 1, 5,13,37/
      data INCORN/ 2, 2, 1, 1, 0, 1, 1, 2, 2/
      data NUMORN/ 6, 6, 4, 2, 0, 2, 4, 6, 6/
      data ORNMNT/-2,     3,
     :            -3,     2,
     :            -1,     3,    -2,     4,
     :            -3,     1,    -4,     2,
     :            -1, 2,  1, 4,  1, 5, -2, 1, -2, 2,  2, 5,
     :            -1,-4, -1,-5,  1,-2, -2,-5,  2,-2,  2,-1,
     :            -1, 2, -1, 3,  1, 5, -2, 2,  2, 5,  2, 6,
     :            -1,-5,  1,-2,  1,-3, -2,-5, -2,-6,  2,-2/
      data HUGE/1000000./,TINY/-1./
C
C     Decide whether or not to ornament at this level
C
      if ( .not.SUCCES(EMBLEV(LEVEL)/float(MLEV1-LEVEL)) ) then
        CNTLEV(LEVEL1) = 2
        EMBLEV(LEVEL1) = EMBLEV(LEVEL)
        PCHLEV(1,LEVEL1) = IPCH1
        PCHLEV(2,LEVEL1) = IPCH2
        return
      end if
      EMBLEV(LEVEL1) = EMBLEV(LEVEL) - 1.0
C
C     Determine form of ornament
C
      I = IPCH2 - IPCH1 + 5
      IPTR = PTRORN(I)
      INC = INCORN(I)
      NUM = NUMORN(I)
C     Assemble schedule of NUM pointers in random order
      do (IORN=1,NUM)
        SCHED(IORN) = IPTR
        IPTR = IPTR + INC
      repeat
      call SHUFLE(SCHED,NUM)
C
C     Select ornament with greatest chromatic interest
C
      CUMO = HUGE
      do (IDXORN=1,NUM)
        CUM1 = TINY
        LPTR = SCHED(IDXORN)
        L = LPTR
        do (INC times)
          CUM1 = amax1(CUM1,CUMDEG(MOD(IPCH1+ORNMNT(L),12)+1))
          L = L + 1
        repeat
        if (CUM1.lt.CUMO) then
          CUMO = CUM1
          IPTR = LPTR
        end if
      repeat
C
C     Pass pitches and update cumulative history
C
      CNTLEV(LEVEL1) = INC + 2
      PCHLEV(1,LEVEL1) = IPCH1
      I = 2
      do (INC times)
        IPCH = IPCH1 + ORNMNT(IPTR)
        PCHLEV(I,LEVEL1) = IPCH
        IDEG = MOD(IPCH,12) + 1
        CUMDEG(IDEG) = CUMDEG(IDEG) + DURLEV(LEVEL1)
        I = I + 1
        IPTR = IPTR + 1
      repeat
      PCHLEV(I,LEVEL1) = IPCH2
```

Ex. 11-1 (page 2 of 3)

```
77      C
78      C     Insert rest
79      C
80            if (SUCCES(RSTLEV(LEVEL))) then
81              J = IRND(INC+1) + 1
82              I = CNTLEV(LEVEL1) + 1
83              CNTLEV(LEVEL1) = I
84              do
85                I1 = I - 1
86                PCHLEV(I,LEVEL1) = PCHLEV(I1,LEVEL1)
87                if (I1.eq.J) exit
88                I = I1
89              repeat
90              PCHLEV(J,LEVEL1) = 0
91            end if
92            return
93            end

 1            block data
 2            parameter (MPCH=5,MDEG=12,MLEV=5,MLEV1=6)
 3            real    EMBLEV(MLEV),CUMDEG(MDEG),DURLEV(MLEV1),RSTLEV(MLEV)
 4            integer TUNE(MPCH),IDXLEV(MLEV),CNTLEV(MLEV),PCHLEV(MPCH,MLEV)
 5            common  LEVEL,LEVEL1,IPCH1,IPCH2,
 6           :        TUNE,IDXLEV,CNTLEV,PCHLEV,EMBLEV,CUMDEG,DURLEV,RSTLEV
 7            data CUMDEG/0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0./
 8            data TUNE/55,53,57,56,60/
 9            data DURLEV/8.,5.,3.,2.,1.,0./
10            data RSTLEV/0.111,0.167,0.250,0.333,0.500/
11            end
```

Ex 11-1 (page 3 of 3)

11.3.2.2 Recursive Processing - The heart of the compositional process is subroutine COMPOS. COMPOS keeps track of the levels of recursion, each of which ornament a melodic fragment passed down from the previous level. The variable LEVEL indicates the level of recursion, array element CNTLEV(LEVEL) indicates how many items occur in a fragment, and array element IDXLEV(LEVEL) indicates which of these items is currently under consideration. For increased economy, COMPOS transfers this value to the holding varible IDX. The pitch of the current item is stored in array element PCHLEV(IDX,LEVEL). For LEVEL=1, COMPOS initializes PCHLEV to the basic tune (lines 13-18), which is specified to the program as array TUNE in line 8 of the BLOCK DATA subroutine. Items at lower levels may be either notes or rests; rests are represented as notes with null pitches. For each item in the fragment but the last, COMPOS performs the following actions:

1. If the current note or rest occurs at the current level of ornamentation, COMPOS directs subroutine WNOTE to append it to the mnemonic listing (line 34 for rests, line 36 for notes). The duration of a note is stored in array element DURLEV(LEVEL) as specified in line 9 of

the BLOCK DATA subroutine; COMPOS selects durations of rests as if the level of ornamentation were increased by one (line 34).

2. If 1) neither the current item nor its immediate successor are rests and 2) the limit (stored in array element EMBLEV(LEVEL)) has yet to be reached, then COMPOS requests subroutine DEDUCE to select a possible ornament at the next level down (line 38) and then advances to the next level (lines 40-41).

The last note in a fragment at the current level is the same as the next note in the fragment at the previous level, so COMPOS backtracks one level (lines 46-47) when IDX reaches CNTLEV(LEVEL).

11.3.2.3 Productions - For each pair of consecutive pitches passed to it either by subroutine COMPOS, or by a prior level of recursion, subroutine DEDUCE first decides whether or not to ornament at the current level. The decision to ornament (lines 24-31) is a yes-or-no trial which incorporates feedback from corresponding trials at higher levels in the manner of the RATIO

(heading 5.3)

feature in Gottfried Michael Koenig's PROJECT2 program. The likelihood of ornamenting is EMBLEV(LEVEL)/FLOAT(MLEV1-LEVEL), where EMBLEV(LEVEL) holds the expected number of ornaments from the current level down while the quantity FLOAT(MLEV1-LEVEL) represents the number of levels (opportunities to ornament) remaining. A decision to ornament causes DEDUCE to reduce EMBLEV(LEVEL) by one before passing it to the next level of recursion. A decision not to ornament causes DEDUCE to skip the remaining actions and return immediately to COMPOS.

If it decides in favor of ornamenting an interval at the current level, DEDUCE must first determine what kind of interval it must ornament and locate a repertory of ornaments for this interval among the eight repertories illustrated in Figure 11-10 (lines 35-44). All ornamental pitches are expressed in array ORNMNT (lines 12-19) as offsets from the initial pitch. For each of the eight intervals, array PTRORN holds a pointer indicating which element of ORNMNT holds the first offset for the first ornament in the corresponding repertory. Array INCORN holds the number of ornamental pitches (1 for escaping or reaching tones, 2 for changing-note groups); incrementing the pointer by this number gives successive ornaments in the repertory. Array NUMORN indicates the number of ornaments in the repertory.

Once it has established a schedule of pointers to each ornament, DEDUCE next applies the methods of the library

subroutine HEUR (heading 7.2) to select the ornament from the repertory which makes the greatest contribution to chromatic diversity (lines 48-61). In order to evaluate an ornamental form, DEDUCE considers each ornamental degree (line 54) to determine how much ~~that~~ the degree has already been used during the composition. In the case of changing-note groups, which supply two ornamental degrees, DEDUCE considers the most-used degree (inner loop: lines 52-56). Among all of the ornaments in the repertory, DEDUCE selects ~~that~~ the ornament whose most-used degree has been used least (lines 57-60). It then converts relative offsets into specific pitches and passes these pitches to the next recursive level (lines 65-76).

As its final action, DEDUCE performs a Bernoulli trial to decide whether or not to include a rest in the ornament (lines 80-91). If it decides favorably, DEDUCE inserts this rest between two consecutive notes in the ornamented fragment.

11.4 DEMONSTRATION 9: AN ARCHITECTURAL GRAMMAR

Like Demonstration 8, Demonstration 9 is hierarchic; however, the hierarchy of Demonstration 9 is architectural rather than functional: beginning with an abstract description of the

composition as a whole, the program recursively applies binary and ternary divisions until, at the terminal level, only descriptions of individual tones remain. The composition reflects Wagner/Lorenz in its superimposition of forms, Tenney in its use of Gestalt principles to project these forms, and both Jones and the author in its implementation.

11.4.1 Compositional Directives

Figure 11-12: Profile of Demonstration 9, measures 1-51.

Figure 11-13: Transcription of Demonstration 9.

The archetype for Demonstration 9 is described by:

1. the duration of the composition (2 minutes),
2. the range of the clarinet, and
3. the twelve degrees of the chromatic scale.

The productions of Demonstration 9 progressively refine this description by dividing sections into subsections and then

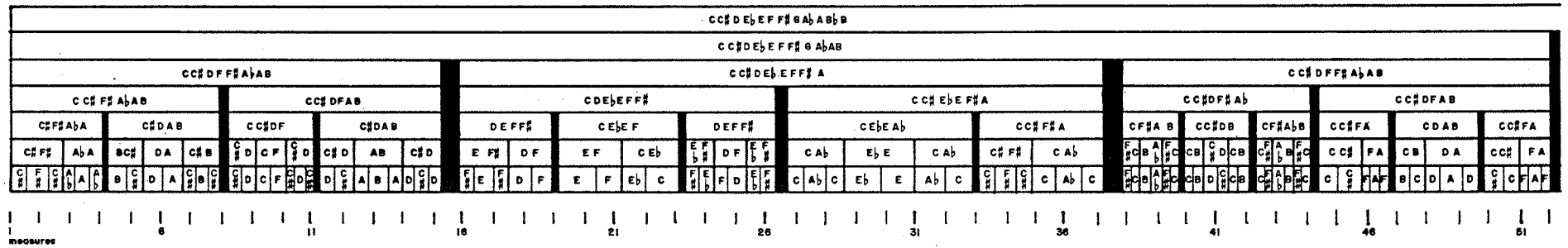


Fig 11-12

11-27a

Demonstration 9

Clarinet

Charles AMES

STRICTLY ♩ = 80

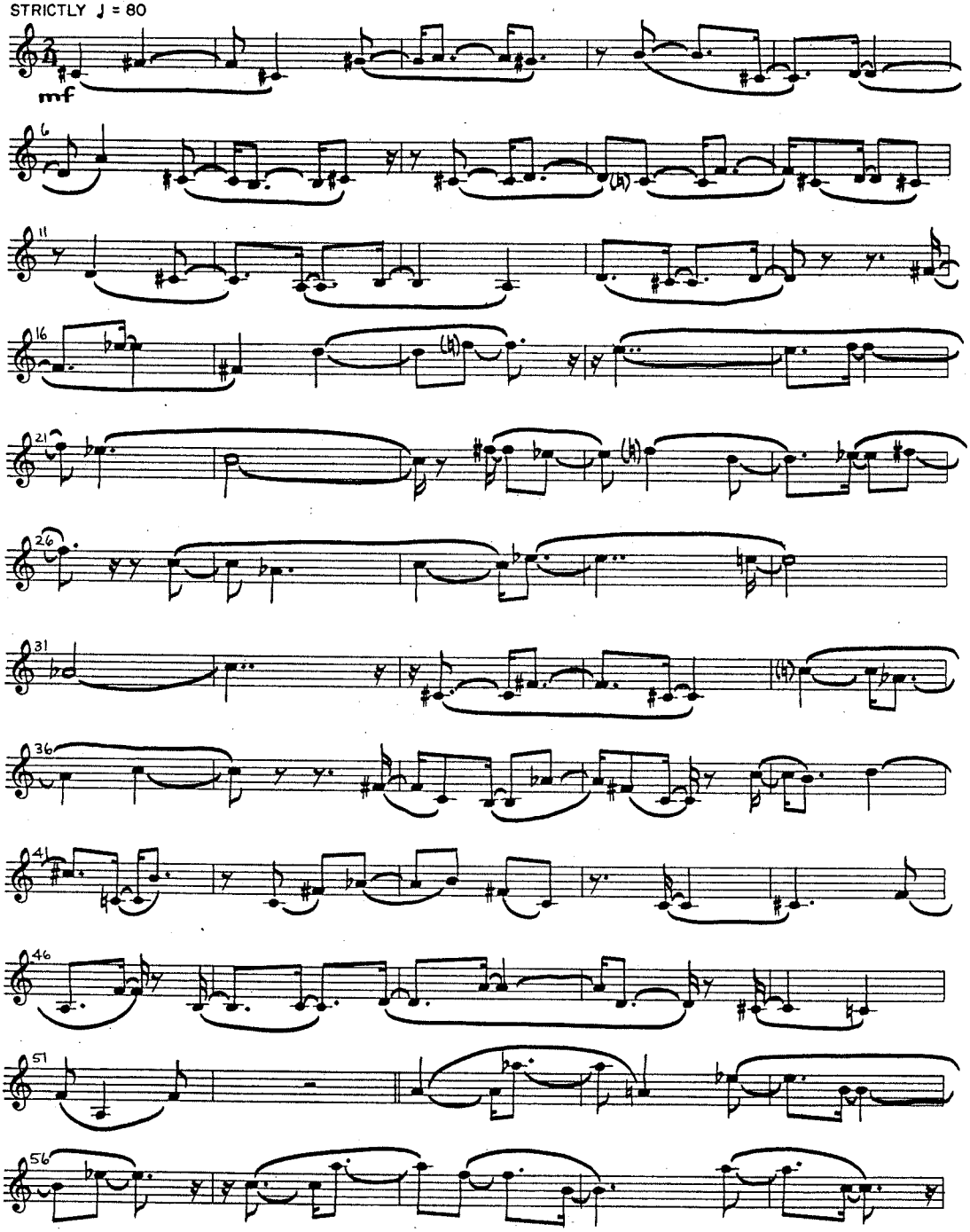


Fig 11-13 (page 1 of 2)

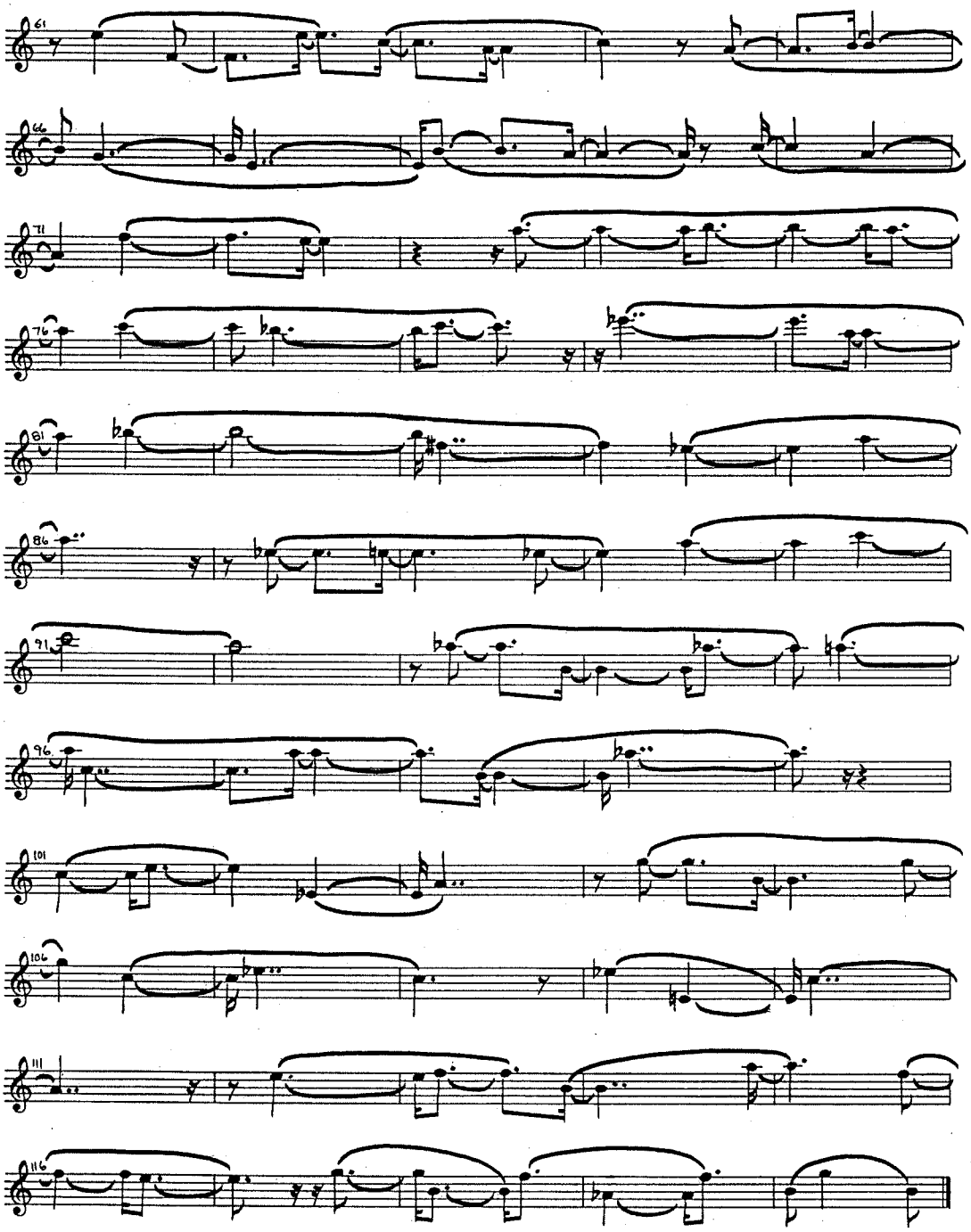


Fig 11-13 (page 2 of 2)

recursively treating each subsection as a section in its own right. Of the two kinds of production,

Binary subsections are asymmetric; A --> a b

Ternary subsections are symmetric; A --> a b a

There are four binary modes and three ternary ones; each specific mode is distinguished by the relative proportions between durations of subsections.

In order to project the grammatic structure clearly, no subsection exploits any registral and chromatic resources not already inherent in the section which produced it. To emphasize similarities between the two outer (a) subsections in ternary divisions, DEM09 passes them identical resources. To emphasize differences between between binary subsections and between (a) subsections and (b) subsections in ternary divisions, DEM09 passes them complementary resources. Each act of division deals with registers by dividing the section's range into thirds, treating the bottom two-thirds as one subrange and the top two-thirds as another. It then passes one subrange to (a) subsections and the other subrange to (b) subsections. With regard to chromatic resources, each act of division derives two distinct (though overlapping) subsets from the set of chromatic degrees characterizing the section. It then passes one subset to

```
      program DEMO9
C
C     Demonstration of architectural grammar
C
      parameter (MLEV=7,MDEG=12,MDIV=7,MCNT=3)
      integer IDXLEV(MLEV),CNTLEV(MLEV),DIVLEV(MLEV),NUMLEV(MLEV)
      integer DEGSCD(MDEG,MCNT,MLEV),CNTDIV(MDIV)
      real    DURLEV(MCNT,MLEV),RSTLEV(MLEV),
     :        RLWLEV(MCNT,MLEV),RHGLEV(MCNT,MLEV)
      real    CUMDEG(MDEG),WGTDIV(MDIV),FACDIV(MCNT,MDIV)
      common  LEVEL,LEVEL1,IDX,CNTLEV,DIVLEV,NUMLEV,DEGSCD,CNTDIV,
     :        DURLEV,RLWLEV,RHGLEV,RSTLEV,CUMDEG,WGTDEG,FACDIV
C
C     Initialize
C
      open (2,file='DEMO8.DAT',status='NEW')
      LEVEL = 1
      CNTLEV(LEVEL) = 1
      IDXLEV(LEVEL) = 0
      do (IDEG=1,MDEG)
        DEGSCD(IDEG,1,LEVEL) = IDEG
        CUMDEG(IDEG) = 1.0
      repeat
      DURLEV(1,LEVEL) = 120. * 8.
      RLWLEV(1,LEVEL) = 40.
      RHGLEV(1,LEVEL) = 68.
C
C     Main composing loop
C
      do
C       Increment section
        IDX = IDXLEV(LEVEL) + 1
        if (IDX.le.CNTLEV(LEVEL)) then
          IDXLEV(LEVEL) = IDX
          LEVEL1 = LEVEL + 1
          if (LEVEL.lt.MLEV) then
C           Write rest or break
            if (IDX.gt.1) call WNOTE(RSTLEV(LEVEL),0,0.)
C           Deduce subsections
            call DEDUCE
C           Advance to next level
            LEVEL = LEVEL1
            IDXLEV(LEVEL) = 0
            WGTDIV(DIVLEV(LEVEL)) = 0.0
          else
C           Write note
            call WNOTE(DURLEV(IDX,LEVEL),DEGSCD(1,IDX,LEVEL),
     :           RLWLEV(IDX,LEVEL)+RHGLEV(IDX,LEVEL))/2.0)
          end if
        else
          if (LEVEL.le.1) exit
C         Backtrack to preceding level
          LEVEL = LEVEL - 1
          WGTDIV(DIVLEV(LEVEL)) = 1.0
        end if
      repeat
      close (2)
      stop
      end
```

```
      subroutine DEDUCE
      parameter (MLEV=7,MDEG=12,MDIV=7,MCNT=3)
      integer IDXLEV(MLEV),CNTLEV(MLEV),DIVLEV(MLEV),NUMLEV(MLEV)
      integer DEGSCD(MDEG,MCNT,MLEV),CNTDIV(MDIV)
      real    DURLEV(MCNT,MLEV),RSTLEV(MLEV),
     :        RLWLEV(MCNT,MLEV),RHGLEV(MCNT,MLEV)
      real    CUMDEG(MDEG),WGTDIV(MDIV),FACDIV(MCNT,MDIV)
      real    FUZDEG(MDEG)
      logical SUCCES
      common  LEVEL,LEVEL1,IDX,CNTLEV,DIVLEV,NUMLEV,DEGSCD,CNTDIV,
     :        DURLEV,RLWLEV,RHGLEV,RSTLEV,CUMDEG,WGTDEG,FACDIV
C
C     Select mode of division into subsections
C
      X = float(MDIV-LEVEL+1) * RANF()
      do (IDIV=1,MDIV)
        W = WGTDIV(IDIV)
        if (X.lt.W) exit
        X = X - W
      repeat
      LCNT = CNTDIV(IDIV)
      CNTLEV(LEVEL1) = LCNT
      DIVLEV(LEVEL1) = IDIV
```

Ex 11-2 (page 1 of 3)

```
      C
      C     Determine duration of each subsection
      C
            DUR = DURLEV(IDX,LEVEL)
            REST = RSTLEV(LEVEL1)
            if (LCNT.eq.2) then
      C       Binary divisions are asymmetric
              DUR = DUR - REST
              MORE = IRND(2)
              LESS = 3 - MORE
              DURLEV(MORE,LEVEL1) = DUR * FACDIV(1,IDIV)
              DURLEV(LESS,LEVEL1) = DUR * FACDIV(2,IDIV)
            else
      C       Ternary divisions are symmetric
              REST = REST + REST
              DUR = DUR - REST
              do (I=1,LCNT)
                DURLEV(I,LEVEL1) = DUR * FACDIV(I,IDIV)
              repeat
            end if
      C
      C     Select registers
      C
            R = (RHGLEV(IDX,LEVEL)-RLWLEV(IDX,LEVEL))/3.0
            if (SUCCES(0.5)) then
              R1 = RLWLEV(IDX,LEVEL)
              R2 = R1 + R + R
              R3 = R1 + R
              R4 = RHGLEV(IDX,LEVEL)
            else
              R3 = RLWLEV(IDX,LEVEL)
              R4 = R3 + R + R
              R1 = R3 + R
              R2 = RHGLEV(IDX,LEVEL)
            end if
            if (LCNT.eq.2) then
              RLWLEV(1,LEVEL1) = R1
              RHGLEV(1,LEVEL1) = R2
              RLWLEV(2,LEVEL1) = R3
              RHGLEV(2,LEVEL1) = R4
            else
              RLWLEV(1,LEVEL1) = R1
              RHGLEV(1,LEVEL1) = R2
              RLWLEV(2,LEVEL1) = R3
              RHGLEV(2,LEVEL1) = R4
              RLWLEV(3,LEVEL1) = R1
              RHGLEV(3,LEVEL1) = R2
            end if
      C
      C     Select degrees
      C
      C     Schedule degrees
            NUM = NUMLEV(LEVEL)
            do (I=1,NUM)
              IDEG = DEGSCD(I,IDX,LEVEL)
              CUMDEG(IDEG) = CUMDEG(IDEG) + REST
              FUZDEG(IDEG) = CUMDEG(IDEG) * RANF()
            repeat
            call DSORT(DEGSCD(1,IDX,LEVEL),FUZDEG,NUM)
      C     Levels 1-5 pass two fewer degrees to levels 2-6
            if (LEVEL.le.5) then
      C       Pass those degrees common to all subsections
              do (I=1,LCNT)
                do (J=1,NUM-4)
                  DEGSCD(J,I,LEVEL1) = DEGSCD(J,IDX,LEVEL)
                repeat
              repeat
      C       Fill remaining positions
              IDEG1 = DEGSCD(NUM  ,IDX,LEVEL)
              IDEG2 = DEGSCD(NUM-1,IDX,LEVEL)
              IDEG3 = DEGSCD(NUM-2,IDX,LEVEL)
              IDEG4 = DEGSCD(NUM-3,IDX,LEVEL)
              if (LCNT.eq.2) then
                DEGSCD(NUM-3,MORE,LEVEL1) = IDEG1
                DEGSCD(NUM-2,MORE,LEVEL1) = IDEG2
                DEGSCD(NUM-3,LESS,LEVEL1) = IDEG3
                DEGSCD(NUM-2,LESS,LEVEL1) = IDEG4
                DUR1 = DURLEV(LESS,LEVEL1)
                DUR2 = DURLEV(MORE,LEVEL1)
```

```
103             else
104               DEGSCD(NUM-3,1,LEVEL1) = IDEG1
105               DEGSCD(NUM-2,1,LEVEL1) = IDEG2
106               DEGSCD(NUM-3,2,LEVEL1) = IDEG3
107               DEGSCD(NUM-2,2,LEVEL1) = IDEG4
108               DEGSCD(NUM-3,3,LEVEL1) = IDEG1
109               DEGSCD(NUM-2,3,LEVEL1) = IDEG2
110               DUR1 = DURLEV(2,LEVEL)
111               DUR2 = DURLEV(1,LEVEL1) + DURLEV(3,LEVEL1)
112             end if
113     C       Update cumulative history for eliminated degrees
114             CUMDEG(IDEG1) = CUMDEG(IDEG1) + DUR1
115             CUMDEG(IDEG2) = CUMDEG(IDEG2) + DUR1
116             CUMDEG(IDEG3) = CUMDEG(IDEG1) + DUR2
117             CUMDEG(IDEG4) = CUMDEG(IDEG2) + DUR2
118           else
119     C       Level 6 passes only one degree to level 7
120             IDEG1 = DEGSCD(1,IDX,LEVEL)
121             IDEG2 = DEGSCD(2,IDX,LEVEL)
122     C       Pass degrees;  update cumulative history for eliminated degrees
123             if (LCNT.eq.2) then
124               DEGSCD(1,MORE,LEVEL1) = IDEG1
125               DEGSCD(1,LESS,LEVEL1) = IDEG2
126               CUMDEG(IDEG1) = CUMDEG(IDEG1) + DURLEV(LESS,LEVEL1)
127               CUMDEG(IDEG2) = CUMDEG(IDEG2) + DURLEV(MORE,LEVEL1)
128             else
129               DEGSCD(1,1,LEVEL1) = IDEG1
130               DEGSCD(1,2,LEVEL1) = IDEG2
131               DEGSCD(1,3,LEVEL1) = IDEG1
132               CUMDEG(IDEG1) = CUMDEG(IDEG1) + DURLEV(2,LEVEL1)
133               CUMDEG(IDEG2) = CUMDEG(IDEG2)
134           :                     + DURLEV(1,LEVEL1) + DURLEV(3,LEVEL1)
135             end if
136           end if
137           return
138           end

  1           block data
  2           parameter (MLEV=7,MDEG=12,MDIV=7,MCNT=3)
  3           integer IDXLEV(MLEV),CNTLEV(MLEV),DIVLEV(MLEV),NUMLEV(MLEV)
  4           integer DEGSCD(MDEG,MCNT,MLEV),CNTDIV(MDIV)
  5           real    DURLEV(MCNT,MLEV),RSTLEV(MLEV),
  6           :       RLWLEV(MCNT,MLEV),RHGLEV(MCNT,MLEV)
  7           real    CUMDEG(MDEG),WGTDIV(MDIV),FACDIV(MCNT,MDIV)
  8           common  LEVEL,LEVEL1,IDX,CNTLEV,DIVLEV,NUMLEV,DEGSCD,CNTDIV,
  9           :       DURLEV,RLWLEV,RHGLEV,RSTLEV,CUMDEG,WGTDEG,FACDIV
 10           data WGTDIV/1.,1.,1.,1.,1.,1.,1./
 11           data NUMLEV/12,10,8,6,4,2,1/
 12           data CNTDIV/2,2,2,2,3,3,3/
 13           data RSTLEV/13.,8.,5.,3.,2.,0.,0./
 14           data FACDIV/.500,.500,0.00, .556,.444,0.00,
 15          :            .571,.429,0.00, .600,.400,0.00,
 16          :            .285,.430,.285, .300,.400,.300, .308,.384,.308/
 17           end
```

Ex 11-2 (page 3 of 3)

CNTLEV(LEVEL) indicates how many sections occur occur in a group, and array element IDXLEV(LEVEL) indicates which of these sections is currently under consideration. The attributes characterizing the current section are described in the following way:

1. the duration of the current section resides in array element DURLEV(IDXLEV(LEVEL),LEVEL);

2. array elements RLWLEV(IDXLEV(LEVEL),LEVEL) and RHGLEV(IDXLEV(LEVEL),LEVEL) hold lower and upper boundaries, respectively, for a value which will ultimately define the lowest pitch in a twelve-semitone gamut; and

3. array element NUMLEV(LEVEL) indicates how many degrees of the chromatic scale are exploited by the current section, while the particular chromatic degrees exploited in this section occupy array elements DEGSCD(1,IDXLEV(LEVEL),LEVEL) through DEGSCD(NDEG(LEVEL),IDXLEV(LEVEL),LEVEL).

For LEVEL=1, DEMO9 initializes these arrays so that at the global level, we may regard Demonstration 9 as a degenerate section (that is, a section with only one subsection). For LEVEL=2

through 6, DEM09 articulates consecutive sections by rests whose durations are stored in array element RSTLEV(LEVEL) (line 38). It also handles all requests to WNOTE at level 6 (lines 47-48).

11.4.2.2 Productions -

The first step in dividing a sections into subsections is to select a mode of division (lines 15-29). Feedback in the manner of Koenig's RATIO feature (heading 5.3) prevents any mode of division from being used more than once along any vertical path one might trace down through the structure. Array element WGTDIV(I) holds the relative likelihood of selecting the Ith mode of division. For LEVEL=1, each element of WGTDIV is set to 1.0, so each mode has equal likelihood of selection. When mode IDIV is selected at level LEVEL, this value is stored in array element DIVLEV(LEVEL) (line 23). WGTDIV(IDIV) is then cleared to 0.0 (line 44 of DEM09) to prevent it from being reselected until the composing process backtracks from level LEVEL, at which point DEM09 resets WGTDIV(DIVLEV(LEVEL)) to 1.0 (line 54 of DEM09).

Array element CNTDIV(IDIV) indicates whether the current mode is binary or ternary; this number is transferred to the holding variable LCNT. The mechanism for selecting durations of subsections varies with LCNT as follows:

1. If the mode of division is binary, then DEDUCE selects the variable MORE to indicate the longer subsection while selecting the variable LESS to indicate the shorter subsection. DEDUCE selects either MORE=1, LESS=2 or MORE=2, LESS=1 with equal likelihood. The array element FACDIV(1,IDIV) holds the relative proportion of the section's duration to be passed to the longer (MOREth) subsection; element FACDIV(2,IDIV) holds the shorter proportion (lines 31-35).

2. If instead the mode of division is ternary, then the middle subsection is always the longest. Array elements FACDIV(1,IDIV), FACDIV(2,IDIV), and FACDIV(3,IDIV) hold the relative proportions of the section's duration to be passed to each subsection (lines 38-42).

DEMO9 next computes ranges for each subsection (lines 47-71). A Bernoulli trial with probability 1/2 determines which subranges go to which subsections.

Allocation of degrees to subsections (lines 76-136) most heavily favors those degrees which have not been heavily exploited elsewhere in the piece. Variables beginning with IDEG indicate chromatic degrees, while array element CUMDEG(IDEG) holds the total duration of all sections in which degree IDEG has

been eliminated up to the current point in the composing process. DEDUCE determines which degrees to pass to which subsections by assembling a schedule in which priorities are based in a "fuzzy" manner (heading 9.2) upon values in CUMDEG (lines 77-81):

1. At levels 1-5, all degrees in the schedule except for the last four are passed to all subsections (lines 86-90). The last four degrees are treated as follows:

   a. If the mode of division is binary, then DEDUCE passes the first two degrees to the longer subsection while passing the remaining two degrees to the shorter subsection (lines 92-95, 97-100).

   b. If the mode of division is ternary, then DEDUCE passes the first two degrees to the (a) subsections while passing the remaining two degrees to the (b) subsection (lines 92-95, 104-109). Notice that durations of (a) subsections, summed together, are consistently at least as large as durations of (b) subsections.

2. The two degrees remaining at level 6 are handled

similarly:

a. If the mode of division is binary, then DEDUCE passes the first degree in the schedule to the longer subsection while passing the remaining degree to the shorter subsection (lines 120-121, 124-125).

b. If the mode of division is ternary, then DEDUCE passes the first degree to the (a) subsections while passing the remaining degree to the (b) subsection (lines 120-121, 129-131).

This process works to balance chromatic resources over the whole composition. DEDUCE completes the feedback loop by increasing CUMDEG(IDEG) for each IDEG eliminated from the Ith subsection by DURLEV(I,LEVEL1), the duration of the subsection.

11.5 NOTES

1. While nested structures are by no means limited to languages, it has been the linguists who have most systematically investigated them. For this reason, we conventionally discuss

nested structures using linguistic terminology.

2. The designations "tune", "rhythmic string", "rhythmic event", "section", and "layer" are the author's. Ashton's original system recognized only the measure as the most general grammatic unit below the composition itself; however, more recent efforts such as the author's implementation (1984 version) and such as the PLA language for musical composition developed by Bill Schottstaedt (1983) incorporate fully recursive facilities for encoding sections and layers.

11.6 RECOMMENDED READING

Ames, Charles. "Crystals: Recursive structures in automated composition", Computer Music Journal, volume 6, number 3 (Fall 1982), p. 46.

Chomsky, Noam. Syntactic Structures (The Hague: Mouton, 1957).

Tenney, James. "Meta + Hodos: A Phenomenology of Twentieth Century Musical Materials and an Approach to the Study of Form", (Inter-American Institute for Musical Research, Tulane

University, 1964).

Holtzman, Steven. "A generative grammar definitional language for music", Interface, volume 9, number 1 (June 1980), p. 1. Holtzman's article includes a summary of Roads (1978).

Jones, Kevin. "Compositional applications of stochastic processes", Computer Music Journal, volume 5, number 2 (Summer 1981), p. 45.