Durham E-Theses

Music analysis and the computer: developing a computer operating system to analyse music, using Johann Sebastian Bach's well tempered clavier book 51 to test the methodology

Broadbent, Clive Graham

How to cite:

Use policy

Music Analysis and the Computer: Developing a computer operating system to analyse music, using Johann Sebastian Bach's "Well Tempered Clavier" Book II to test the methodology.

Clive Graham Broadbent

"Most computerised and computer-aided musicological projects are written to achieve specific goals. Once achieved or not achieved as the case may be, the projects and their tools are frequently discarded because their dependency upon specific computer hardware and software prevents them from being utilised by other researchers for other projects. What is needed is a system that, using small tools to accomplish small tasks, can be expanded and customized to suit specific needs.

This thesis proposes the creation of a music-analysis computer operating system that contains simple commands to perform simple musicological tasks such as the removal of repeated notes from a score or the audible rendition of a melodic line. The tools can be bolted together to form larger tools that perform larger tasks. New tools can be created and added to the operating system with relative ease, and these in turn can be bolted onto old tools.

The thesis suggests a basic set of tools derived from old and new analytical methods, proposes a standard for their implementation based on the UNIX computer operating system, and discusses the benefits of using the system and its tools in an analysis of the twenty-four fugues of Johann Sebastian Bach from the "Well Tempered Clavier", Book II."

I confirm that no part of the material offered has previously been submitted by me for a degree in this or in any other University.

Signed   ....

Date   ....

Music Analysis and the Computer: Developing a
computer operating system to analyse music, using
Johann Sebastian Bach's "Well Tempered Clavier"
Book II to test the methodology.

Clive Graham Broadbent

Master of Music
University of Durham
Department of Music

1994

i

Contents

Introduction

Only those who have had the singular pleasure of caring for young children are aware of the innocent and yet unanswerable questions which pour out from their naïve mouths. The 'why' questions which sound so short and simple, and seem to beg for equally short and simple answers, appear only stoppable with book-length explanations. So often a vicious 'question-answer' circle evolves when frustrated parents attempt to ward off these 'simple' questions with their equally 'simple' 'because' answers.

The purpose of this research is to procure a novel approach to computer-aided music analysis, to provide a useable mechanised-system which will help answer the many questions arising from the perusal of music scores or the hearing of musical performances. The new approach will not, however, attempt to provide the 'correct' answer for a specific question, but will aim instead to help a user extract information from a music score which might be of relevance during the formulation of his or her own answer. It is hoped that an analytical method, system, or environment can be created which offers the versatility of intuitive manual analyses and yet has a set of underlying mathematical rules or techniques which enable it to be created for use with a computer. A manual analysis which might normally take days or even months could be reduced to a matter of hours or even minutes, and perhaps provide new answers to old and familiar questions.

Mark Ellis, when questioning the use of statistical methods of analysis in music, "...hoped that the increased use of [statistical] methods [would] enable great progress to be made in the study of the mind of a composer".[1] Likewise, it

---

[1]   Ellis, M. R., "Are Traditional Statistical Methods Valid for Musical Analysis?",
       Proceedings of the Second International Symposium on Computers and Musicology,
       Orsay, 1981 (Paris, CNRS, 1983), p. 194

is hoped that use of, and expansion of the Analysis Environment proposed in this thesis will also help unravel some of the mysteries behind the subjective views of success and failure in composition.

Music is complicated. Designing a computer system to answer all questions would undoubtedly take forever and be so complicated that the programming would probably never be completed. "There is a danger, however, that the system may become the end in itself, taking so long to design and build that it is obsolete before it produces significant results".[2] To circumvent these problems, the proposed Analysis Environment contains a number of ready-made computer tools to perform specific analytical tasks with musical compositions, such as removal of repeated notes, determination of key, or identification of similarity.

Some of the tools appear to perform simple tasks, tasks which are not complicated manual operations, but tasks which become prone to error and tedium when applied to lengthy music data. Such tools are primarily designed to reduce the likelihood of errors, and enable the analyst to devote his or her time to the other perhaps more interesting aspects of the music data. These simple tools can be adjusted either to perform their role using an unusual methodology or to yield their output in an unusual format, providing new angles on the human process.

Modelled on the UNIX computer operating system, the Analysis Environment allows complete expansion whereby a new tool can be created with the minimum of fuss, to perform a specific task, and added to the environment. The new tool can be joined to other tools and exploited to procure results to more complicated problems. A bountiful supply of tools, created by

---

[2]   Erickson, R., "Music Analysis and the Computer", Journal of Music Theory, Yale School of Music, Vol. 12, No. 2, 1968, p. 260

researchers to answer their own particular problems, might even provide the foundation for a group or society that interchanges the very latest tools, rather like UNIX, where programmers submitted their favourite tools to the University of California, Berkeley, and the most useful were included in the next release of the UNIX operating system.

"One cardinal principle guides our evaluation of the data provided for us by the computer: never to be deluded into believing that the computer was in any way capable of analyzing a piece of music".[3] At no stage will the computer actually analyse the music from within the proposed environment. The tools of the environment will merely extract, transform or translate the information in an encoded version of a music score to provide the researcher with a different form of the information to which he or she can apply his or her own analysis. Such a system can be regarded as computer-aided analysis as opposed to computerised analysis. A 'black-box' system where users are unaware of what is going on inside and only ever see the final results of their mystical 'push-button' analyses will drive away the sceptical. A system that uses small, uncomplicated tools to perform small uncomplicated tasks, however, will be attractive to the researcher and encourage him or her to bolt them together in a "Meccano"-like fashion to achieve a desired result. Like "Meccano", the builders or analysts are not restricted to the instruction manual and its limited set of models or analytical methodologies. Experimentation is encouraged. Assembling arbitrary flanges and plates to see what might result is analogous to attaching tools to see what comes out.

---

[3]   Bernstein, L. F. and Olive, J. P., "Computers and the 16th-Century Chanson: A Pilot Project at the University of Chicago", Computers and the Humanities, 3 (1968-1969), p. 160

Such a system, together with the necessary human intervention and cooperation, will produce rather more meaningful and worthwhile results than those manifested by 'something' else.

UNIX is the operating system many believe will take over the world. Initially, UNIX and its source code was distributed at a nominal cost to universities, encouraging its customisation and the development of many new tools. Perhaps similar publicity and distribution of the proposed Analysis Environment will achieve the same amount of interest and growth? UNIX's simplicity of expansion is one of its strongest features—any researcher able to program a computer in a high level language such as Pascal[4] or C[5] can add new tools to the operating system, and, like UNIX, the researcher will encounter few difficulties when creating new tools and adding them to the proposed Analysis Environment.

A computer operating system is a collection of programs which control the overall operation of a computer. The proposed Analysis Environment, therefore, since it is designed to harness the power of a computer for the sole task of analysing music, might also be regarded as an operating system. Operating systems generally allow a user to carry out a variety of tasks using a small number of commands. With an operating system geared towards music analysis, musicologists have a 'nut-cracker' which can be harnessed in a variety of ways to crack a variety of 'nuts'. So often, musicologists find a 'nut' to crack, and then build the 'nut-cracker'. Most musicologists, firmly grounded in the academics of music and most likely only ever having acquired their

---

4   Pascal, named after Blaise Pascal (1623–1662), is an easy to learn programming language which supports the concept of structured programming, and follows a precise form.

5   C is a general-purpose programming language which can be programmed at a high level (using english-like commands) or a low-level (allowing direct access to the computer's memory and data).

computer programming skills through self-tuition, are almost always reluctant to share their work with others in the fear that their programming might not be 'up to scratch'. A situation has now arisen where many analytical programs exist, but they are, since their creation was for large specific tasks, of limited used, difficult to obtain and probably incompatible with different forms of data. An operating system or environment would allow the interchange of tools, techniques, data and ideas and provide a 'nut-cracker' rather than a 'cracked-nut'.

Tracing a path through the development of computer-aided musicology reveals many interesting and varied experiments. Many of these experiments, projects and methods have been worthwhile. Some, sadly, have been disappointing, even to the point of failure. The majority, successful or unsuccessful however, cannot be used again due to their dependency upon either specific computer hardware, software or even music encoding language. Despite this, perhaps some of the techniques can be incorporated into the proposed Analysis Environment? One goal of this research is to ascertain what can and cannot undergo straightforward conversion and be utilised as a tool in the Analysis Environment, eventually evolving into a standard computer-aided analysis environment.

The Analysis Environment, however, is not solely the domain of tools which adhere to previously conceived techniques. One of the major banes in computerising music analytical methodologies has been that of quantifying similarity. How can one instruct a computer to ascertain the similarity between two melodies? The formulae for a tool which determines similarity was derived from acoustic theory, and the development of this and other specific analytical tools are covered in depth within this thesis.

Dictionaries define analysis as the resolution of an object into simpler elements, with the intention of finding and showing the object's structure. The

5

elements of musical structure are well understood by musicologists, and analysis for the trained musicologist is not simply the resolution of a score into elements, but the examination of element fusion—the successful way in which elements are combined to make 'appealing' music. Effective fusion creates an energy which lasts forever, and ineffective fusion yields nothing. The less expert musicians and academics might be aware of the smallest musicals elements—atoms, but not the mid-level elements made up of atoms. For them, the first stage in an analytical process which will educate them is to determine the low-level elements and then the mid-level elements. The tools of the Analysis Environment make the process of 'resolving' a music score into low and mid-level elements more straightforward for the inexperienced musician. For seasoned musicologists, successful re-synthesis of music from low and mid-level elements will not only prove that their anatomisation of elements was realistic, but also that their analytical methodology was effective. One might argue that more information can been gained from the re-synthesis process than from the de-synthesis process. The Analysis Environment, however, contains simple tools to operate either way. The intention of all such procedures is to learn more about the structure of a composition. The Analysis Environment, therefore, must surely be a step nearer towards useful distributable computer-aided music analysis.

Akin to those apparently unanswerable 'Why is grass green?' questions, innocently spewed out by naïve and inquisitive children seeking to understand the world, Pinkerton, in his 1956 article entitled "Information Theory and Melody",[6] set the following poser: "What is it about simple melodies that makes them so widely appealing?".[7] Such an interesting topic provokes further investigation. What is a simple melody? Is simplicity the major appealing factor? Where is the cut-off point between appeal and boredom when simplifying a melody? The original question itself, however, is no more extraordinary than those of other musicologists who crave for the same insight into the secret of melody. Pinkerton, however, employed mathematics in his quest for the answer. Integrating mathematics and music, although unusual, was nothing new, since Joseph Schillinger[8] had also tried to determine what "...makes eyes light up and sets feet tapping...".[9] Here, however, Pinkerton was attempting to recreate, with the aid of his 'banal tune-maker', human thought processes.

Running parallel with the increased use of mathematics in musicology, was the rapid development of computer hardware and programming techniques. Dr. Martin Klein, in a 1956 address[10], stated that "The computing

---

[6]   Pinkerton, R. C., "Information Theory and Melody", Scientific American, 194 (Feb., 1956) pp. 77-86

[7]   Pinkerton, R. C., op. cit., p. 77

[8]   Schillinger, J., The Mathematical Basis of the Arts, Reprint ed. (New York and London: Johnson Reprint, 1966). Schillinger used statistical data to measure consistency of style in the works of major classical composers, ie two pieces by the same composer might be consistent in style if the number of key changes is the same, etc. From his analyses, he produced a set of compositional rules.

[9]   Pinkerton, R. C., loc. cit.

[10] Klein, M. L., "Uncommon Uses for Common Digital Computers", Instruments and Automation, 30 (1957) pp. 251-253

machine designer has one objective—to construct a digital computer which occupies no space, uses no power, and performs a thousand or so distinct operations on a memory that holds an infinite number of words",[11] and thus, with musicological problems becoming more and more complex, it is not surprising that eventually the two seemingly diverse fields of computers and music began to merge.

Since a musical score contains informative elements (such as pitch and time) which can be accurately measured, Hiller stated that music, in the form of a score, was "accessible to rational and ultimately mathematical analysis".[12] Computers, although gradually becoming everyday objects, were still viewed very much as 'magic boxes' which, to the 'man-in-the-street', were capable of anything which one could imagine. As always, when faced with a 'magic box' that will solve anything, one is inclined to ask it to perform the most incredible task which one can devise, as opposed to a straightforward menial operation.

Many musicologists are only too willing to devise new analytical theories and methodologies. Sadly, such theories are normally only ever tested via laborious and painstaking manual analyses which are frequently prone to error—errors arising from miscounting of notes, misreading of scores or simply lapses in concentration.

It was several years after the publication of Hiller and Isaacson's "first book presenting the application of scientific method to composition"[13] before musicologists, such as W. S. Collins, began to realise the full potential of using computers for analytical aid.

---

[11]   Klein, M. L., op. cit., p. 253

[12]   Hiller, L. A. Jr., "Computer Music", Scientific American, 201 (Dec., 1959) p. 110

[13]   Hiller, L. A. Jr. and Isaacson, L. M., Experimental Music: Composition with an Electronic Computer, McGraw-Hill Book Co., New York, 1959, (flysheet)

The apparently misspelt word 'program' was still fighting for its rightful place within the English dictionary. Artistic acceptance was the only encouragement computer programmers needed to tackle the problems of music theory. "Experimental Music" received numerous reviews, and not simply because Hiller was 'plugging' it wherever and whenever he could. The academics were curious. Even The Baldwin Piano Company managed a small report. Not all reviews, however, were patronising and favourable. P. A. Evans, in reference to the "Illiac Suite", stated, "Any teacher would recognize it as the work of a determined but singularly unmusical pupil".[14]

Other professed achievements were well in hand. Walter S. Collins, believing himself to be one of only a handful of musicologists to sense the potential for computer aid in musicology, realised that searching for self-quotation within the works of Thomas Weelkes (c. 1575–1623) could be adapted for computer. Weelkes would frequently borrow musical material from one composition to use in another. However, considering himself to be incapable of writing computer programs, Collins declared that programming "would probably be outside the competence of the average musicologist"[15] and had to draft in extra help to create an 'analysis' program which detected identical repetition of six-note segments within, and between different compositions.

Collins' interest was in Weelkes' church music, and the seemingly mammoth task of squeezing his entire corpus of church music into a computer was made more manageable via a selective process. Only the bass lines of thirty-four canticles were eventually input to the computer—based on a

---

14   Evans, P. A., "Reviews of Books: Experimental Music", Music and Letters, 42 (1961), p. 370

15   Collins, W. S., "A New Tool for Musicology", Music and Letters, XLVI (1965), pp. 122-5

knowledge of Weelkes' style, discovery of bass-line repetition would most likely reveal repetition in the higher parts as well.

When translating the canticle bass lines into an alphabetic code which could be stored with relative ease within computer memory, key signatures, separate accidentals, and rhythm were ignored. Notes of the scale were encoded using letters of the alphabet and, during processing, these notes were converted into ascending or descending intervals—allowing detection of repetition in different keys or modes. Collins' program discovered quotations which hitherto had not been recognised, and confirmed that a computer could indeed be of value in musicological research. With the prevalence of scientific ignorance in an artistic world, Collins went so far as to quote a one-hundred-and-thirty segment-per-second rate achieved by an IBM 1620 computer—perhaps as bait for the sceptical. Musicologists were certainly becoming aware of the advantages of computer aid. Clerical tasks such as compilation of indices, bibliographies, counting second-inversion chords and so on, which "result in immense lists of statistics which prove little but the persistence of the author",[16] were no longer a chore; and more complex tasks such as solving puzzle canons, completion of works surviving incomplete, and identification of anonymous works appeared within reach. Collins suspected that nearly all facets of a composer's style could be "translated into a computer programme" and thus studied using similar methods to that of his self-quotation analysis program.

For almost a decade, music research had accounted for only a small percentage of general computer use. Like the gradual discovery of new limbs and a knowledge of how to use them, computer ignorance and innocence slowly disappeared, and the crawling toddlers of computers and music research

---

[16]   Collins, W. S., op. cit., p. 122

began to walk. Many large and ambitious projects emerged. All the masses of Josquin Desprez were being methodically typed into computer with a view to future stylistic interrogation, and Milton Babbitt was so confident in the forthcoming success of the "Josquin Project" that he declared, "questions of intervallic succession and simultaneity, correlations between text and music, decisions as to matters of 'ficta', etc should be forthcoming very soon".[17]

Michael Kassler started transferring Heinrich Schenker's analytical theory onto computer—but it still required testing; and other scholars produced specialist lists of hexachords and complete lists of all-interval sets. Despite the benefits to be gained from sharing acquired knowledge, the secretive and guarded approach to research by many scholars led to a duplication of research in numerous areas; and, independent lists of all-interval sets were generated by Stefan Bauer-Mengelberg, Melvin Ferentz, Hubert Howe, and Eric Regener to name but a few. This lack of knowledge-sharing was marginally offset by the publication of a handful of short papers and articles endeavouring to give an overview of computer usage in humanistic research.

Since mathematical analytical theories adapt more readily to computer than those theories which are essentially intuitive, a large proportion of computer-aided music research investigated the properties and relationships of pitch-class sets. Attempting to "make the best possible sense out of a composition...",[18] Hubert S. Howe spent time experimenting with sets containing less than twelve pitch-classes. Unlike the texts written by the uncommunicative researchers, the article written by Howe[19] gave both an

---

17   Babbitt, M., "The Use of Computers in Musicological Research", Perspectives of New Music, 3 (1965), p. 74

18   Howe, H. S., "Some Combinational Properties of Pitch Structures", Perspectives of New Music, 4 (1965), p. 59

19   Howe, H. S., op. cit., pp. 45-61

historical background and a general survey of contemporary exploration. Howe's generosity in distributing information extended to include a FORTRAN program in the appendix (for calculating pitch structures of size ten to eleven), and an offer to provide useful software for any interested parties. Even with the realisation that any analytical intentions have to be stated precisely and clearly for an accurate computer implementation, computer limitations often lead to an adaptation of one's original analytical theory, and produce the egg and chicken syndrome—which came first, the theory or the computer?

Collins' theory came first. His success was largely due to the total conception of an analytical method before the discovery of a computer's capabilities. If his naïvety had been lost through an early meeting with computers, computing power (or lack of) would have greatly influenced the development of any analytical theory. Unlike Collins' fresh approach, John Selleck and Roger Bakeman experimented with computers first, and then searched for a task in which to employ the computer. Selleck and Bakeman used data-processing techniques to free themselves from arduous tasks. Unlike the carefree teenager who gaily tosses a coin into an arcade machine before reading any instructions, and then loses both time and money when a lack of knowledge brings an early end to the game, Selleck and Bakeman discussed their work and then prepared themselves. An analytical aim was proposed. Their FORTRAN II programs, although never fully developed and only designed for simple data (Gregorian Chants), identified identical repetition, constructed melodies in Mode Four,[20] and defined melodic patterns which occurred in more than one context. Since both the length and repetition of notes are often dependent upon chant text, their simple coding of the chant

---

[20] Mode Four may be regarded as a scale comprising the 'white notes' of the piano or organ, ascending from B to B. Likewise, Mode Five comprises the 'white notes' ascending from F to F.

ignored rhythm and note repetitions. The actual analysis (entitled 'phrase identity') in the first of a set of three programs, compared every 'phrase' (a term never actually defined) with every other phrase to yield identity or non-identity. The results, assembled and formed into a probability table, provided the basis for a second program which generated Mode Five (see previous footnote) phrases—an exercise in already developed techniques. Like so many previous attempts at phrase-generation based merely upon transition probabilities, only a handful of their generated phrases resembled those in the analytical sample. Program three, progressing ever-closer to actual analysis, attempted to determine which 'melodic cells' (patterns which appear in numerous contexts) were employed in phrases. Melodic cells were of interest if they appeared only once and were not subsets of larger melodic cells. Realising, like others, that computers are only very fast and very accurate, and will not accomplish anything other that what they are instructed to carry out, Selleck and Bakeman did not attempt to fill their IBM 709 with all the analytical theories of the decade. Since their analytical results were not intended to provide 'the answer', a fresh insight into the material under analysis was gained, and further analytical procedures were developed upon perusal of their results. Many musicologists, upon the discovery of a mysterious box which can quickly draw graphs on a visual display and perform complex mathematical calculations without effort, imagine that a computer implementation of an analytical theory is there for the taking by anyone who has the patience to convert such a theory into mathematical algorithms. However, the 'box' does not respond to the command 'analyse this piece'. What does 'analyse' mean? What does 'piece' mean? Such general terminology seems indefinable in English, let alone in logical and mathematical terms. 'Analysis' with computer, if it is to be regarded as more than identical pattern-matching or statistics-counting, could be redefined as 'a reworking of material to provide new insight into the material's structure'—which is what the research of Selleck and Bakeman was gradually moving toward.

One of the first conferences, actually tackling the controversial subject of computers and music, took place on December 4th, 1964. However, it was not until 1966 that, through a review by Allen Forte,[21] the contents of the conference became known to scientists and musicians alike. Acting for the defence in the fanciful trial of 'computer murders music', Stefan Bauer-Mengelberg was quick to point out that as well as historical and literary-type problems, musicologists faced special problems of their own. Barry Brook supported the concept of computers used for bibliographical research, whilst George Logemann discussed the problems faced in music-data representation and related the success of his method for solving puzzle canons. However. it was Michael Kassler's description of his project—"...the use of a generative grammar, in the Chomsky sense, for the explication of Schenker's theory of music structure"[22]—which finally tipped the balance and extracted a 'not guilty' verdict from a more-than-likely 'pro-computer' jury.

Kassler's main interest though, was in the extraction of information from musical data, and his programming language MIR (Musical Information Retrieval), designed as part of a larger project, allowed examination of particular syntactic attributes within a composition. Kassler claimed that his system was not only comprehensive, "...any predicate whose truth-value is computable from the notes, rests, clefs, and other 'primitive symbols' of musical notation that in some order constitute one or another particular composition—can be represented as a program in MIR",[23] but also a foundation and starting point for others to create alternative and more advanced systems, "I have

---

21   Forte, A., "Conference on the Use of Computers in Humanistic Research: A Review", Computers and the Humanities, 1 (1966-1967), pp.110-112

22   Forte, A., op. cit., p. 112

23   Kassler, M., "Toward Music Information Retrieval", Perspectives of New Music, 4 (1966), p. 59

spoken of MIR at such comparative length that I might give you a good idea of what I believe is the most advanced existing system for musical information retrieval, and that I might provide, therefore, a realistic base to which proposers of better systems could refer".[24] The MIR language was not just another 'proposal' since Tobias Robison, with help from Hubert S. Howe, Jr., and Kassler, had managed to write a computer program, on an IBM 7094, which automatically carried out any ordered set of well-formed MIR instructions. Example programs and explanations even reached journals and papers, allowing the whole world to share in their innovation. However, there were limitations with this 'comprehensive' language. Musical compositions (for processing) had to exist in a 'lynear partition'[25] —where every note in a partition could be performed by an instrument which could play no more than one note—and thus, polyphonic analysis was out of the question. Each line (or one punch card) contained a single MIR instruction, and during the execution of a program, only one note or rest in the musical composition being processed would be examined at any instant in time. Therefore, simple data extraction was relatively straightforward, whereas comparison of different musical sections within a composition was no more, if not less effective than a manual comparison.

Allen Forte, like many others, also believed that a faithfully and successfully encoded composition could be analysed using a computer program—"The musical score... constitutes a complete system of graphic signs and, properly represented for computer input, may be analysed by a program as a logical image of the unfolding musical events that make up the

---

24   Kassler, M., ibid.

25   Kassler was trying to develop his own concepts of "lyne" and "lynear". If a melodic line can be played on an instrument which is only capable of producing one pitch at any one time, the melodic line is deemed to be a "lyne". A "lynear" partition is made of "lynes".

composition".[26] Whilst holding a fellowship in the American Council of Learned Societies (sponsored by IBM), Forte developed a computer program for the analytical reading of scores. Although information about a specific composition may be gained from reading descriptions of a composer's techniques and views or opinions, or by obtaining descriptions from listeners, the program was designed instead to extract information (possibly not available from the aforementioned methods) from the score itself. Since even human analysts make some of their decisions following rules, perhaps, therefore, it would not be wrong to convert some of their rules into computer algorithms? Forte's interest in atonal music led him to transfer set analysis rules and techniques into algorithms for use in computer analysis. Music data was encoded by hand using a language designed by Stefan Bauer-Mengelberg[27] (although a simplified form was employed during program development) and analysed by a SNOBOL program (a programming language which handles free-form strings) which simply consisted of a number of functions. The encoded data and program allowed one "...to refer to any moment in a composition and to relate any event to any other event with respect to time-continuum".[28] Initially the program scanned instrumental parts to extract 'primary segments' (monody delimited by rests) and inserted implicit accidentals. 'Secondary segments' resulted from the interaction in time of the attacks and releases of two primary segments in different instrumental parts. Pitch classes were then taken from the primary and secondary segments, and arranged in ascending order with duplicates removed, to produce 'compositional sets'. Forte's analysis program

26   Forte, A., "A Program for the Analytic Reading of Scores", Journal of Music Theory, Vol. 10, No. 2 (1966), p. 332

27   DARMS, "Digital-Alternate Representation of Musical Scores". A full description of DARMS may be found in Raymond F. Erickson, "DARMS, A Reference Manual" (New York: Queens College, CUNY, 1976)

28   Forte, A., op. cit., p. 341

was purported to determine the class to which each set belonged, list and count all occurrences, calculate set-complexes, and retrieve historical and informal comments, to name but a few of the many functions. Even with so much to offer, and the 'advertising' via Forte's article in the Journal of Music Theory, there was still, in Forte's eyes, much work to be completed.

Nineteen months after the 'Conference on the use of Computers in Humanistic Research', the first working seminar on the application of computer technology to musical problems took place at the Harpur College, in The State University of New York at Binghamton. The twelve-day seminar provided tuition, via lectures and copious reading of manuals, in score encoding (using DARMS), computer programming with languages such as SNOBOL and FORTRAN, and some practical experience in writing and executing a program (even to the extent of lessons in punching cards). The seminar intended to spread practical knowledge of a limited field, and had long-term goals, with perhaps dreams of students producing computer systems to cut time-consuming routine labour or eliminate duplication of effort through bibliographical control.

Like a 'gin and tonic', where two entities exist successfully in isolation yet appear so compatible that when merged one wonders how the two were ever kept apart, the phrase 'Musicology and Computers' was no longer turning heads and furrowing brows. By 1966, a session devoted to 'Musicology and the Computer' had crept into the American Musicological Society Annual Meeting. The 1966 session contained three papers, each describing projects in their early stages of research. Two of the three researchers, Professor Harry Lincoln and Professor Lawrence Bernstein, chose to examine vast quantities of material with a view to creating large-scale concepts, adaptable methods, and some general results. Professor Roland Jackson, on the other hand, restricted himself to only a small amount of data, allowing more time to consider minute

details. Lincoln's paper, "The Frottole Repertory: A Pilot Study in Information Retrieval", concerned itself mainly with the problems of concordance hunting. A simple program written in any string manipulation language (eg SNOBOL) would compute numerous 'list' and 'compare' operations to locate music-data, and provide accurate and rapid printout results via recently developed Xerox semi-micro cards which had relevant data recorded onto them.

Jackson, however, was rather more ambitious, and his paper "Harmonic Analysis with Computers as Applied to Contemporary Music" discussed his intentions, following the reduction of each vertical interval in a piece to an integer, to compute a vast variety of problems relating to contemporary works, including: frequency distribution of chords in their inversions, breakdown of the number of pitch classes sounding, a dictionary of chords used, analysis of free-standing tones and the voice in which they occur, comparison of dissonance, and analysis of recurrent chord patterns and recurrent individual chords of varying types. Needless to say, with all these many tasks, Jackson's research had reached no general conclusions at the time of the 1966 meeting.

"Problems of Stylistic Analysis of the 16th-Century Chanson" revealed Bernstein's attempts at computerised style analysis. The stylistic criteria most frequently employed in analysis of 16th-century chansons, such as texture, melody, rhythm and so forth, were to be determined automatically via a system of encoding and a series of analytical programs grouped under the title 'Chicago Linear Music Language' (CLML). Marian Cobin's review of the meeting summarised the prime goals of computer-aided research: "...the ability to look at a work of art in its entirety and in all its myriad detail, to correlate data with mechanical aid, and then armed with the greater battery of information, computer-stored and computer-correlated, to evaluate the work—first on its own merits, then within the composer's oeuvre, within the period and genre of

the work, and finally within its place in cultural history".[29] Bernstein's paper reminded the audience about these important goals.

Success at creating a computer program, to aid transcription of mid thirteenth-century polyphonic notation, aroused Maurita and Ronald Brender's interest in other areas of computer-assisted musicology, and the Brenders steered their future research toward analysis. Another computer program performed analysis on five motets with Portare tenors from the "Bamberg Codex". The first stage of the analysis recognised and recorded occurrences of part or voice crossings to determine to what extent the motetus, tenor, and triplum could be associated with a fixed position in the chord structure. The second stage recorded and tabulated all melodic intervals found in each part to produce an 'average rate of movement'. This rate of movement, together with other results showed that descending lines tended to change or move faster over larger intervals than ascending lines. Since computer programs were used to help test the validity of notions not previously checked carefully and provide new insights into the music, the Brenders felt that their use of computers for analysis was justified.

In order to keep the Humanities knowledgeable about the world of computers, Thomas E. Binkley was kind enough to include a short definition of computer terms in his article entitled "Electronic Processing of Musical Materials".[30] His explanation of terms and some computer concepts progressed gradually towards the practice of storing music data in computer words or addresses and the ability to compare such words to assess a degree

---

29   Cobin,. M. W., "Musicology and the Computer in New Orleans", Computers and the Humanities, 1 (1966-1967), p. 133

30   Binkley, T. E., "Electronic Processing of Musical Materials", Elektronische Datenverarbeitung in Der Musikwissenschaft, Harald Heckmann, ed., Regensburg: Gustav Bosse Verlag, (1967), pp. 1-20

of identity where "The musical relationship, or degree of sameness is a reflection of the musical relationship".[31]

Binkley's article contained a method for comparison of incipits, and even considered the comparison of entire pieces to be possible—so long as they did not contain polyphony. Binkley's method is straightforward. A unique address or word of computer memory is set aside for each pitch available, whilst the bit pattern stored in an address represents the occurrence of that pitch throughout the incipit. Each bit in an address signifies a sixteenth-note duration (or perhaps any value a programmer wishes). Thus, if the memory address set aside for middle C contains the bit pattern "1111000011110000", a middle C occurs on the first and third crotchet beats of the incipit.

The comparison of incipits is simply a matter of comparing bit patterns. Each incipit comprises a series of bits—a bit can be set to one or zero. When comparing two incipits, the series of bits for the first incipit is placed above the series of bits for the second incipit. Each of the first incipit's bits is compared in turn with the equivalent bit in the second incipit's bit series. If the current bit of both the first and second incipits is set to one, a positive counter is incremented by one. If the current bit of both the first and second incipits is different, a negative counter is incremented by one. The degree of sameness is calculated using a simple formula where minimum sameness equals zero, and maximum sameness equals one. For Binkley, a degree of sameness of 25/40 was high enough to warrant interest in both of the compared pieces.

Although similarity testing was now being used for incipits in thematic catalogues, no one had entertained the notion of using it for full-blown analysis. Determining the chant source, for example, of a given melody is usually a

---

[31]   Binkley, T. E., ibid.

laborious task, often involving a manual search, page by page. As Binkley said, "How much longer are we going to continue in serene folly, pretending that busy work is scholarly work?".[32] Murray Gould proposed that a set of music data (the contents of the "Liber Usualis") could be encoded using a recently developed encoding language (ALMA—Alphanumeric Language for Music Analysis) and stored on magnetic tape or disk. The search for a melody would then be reduced to a matter of minutes, no matter what key or mode the melody was in. Busyness might then perhaps become scholarlyness. Gould, like Binkley, raised the concept of algebraizing similarity by suggesting that a criterion of similarity could be specified in the computer program designed to execute the searching, and thus a search would also produce close variants of a melody.

"Fear of mechanism is one of the humanist's soundest instincts. ...it tends to close the minds of some humanistic scholars to potentialities of modern technology that can free them of deadingly repetitive bibliographical tasks, provide new means of communicating their ideas, increase accuracy in dealing with source material, and even stimulate new directions of thought".[33] Jan LaRue and Marian Cobin were fully aware of the advantages that computer-aided musicology could bring.

As in most historical periods, nothing could stop the destined interaction between music, science and mathematics. Even seemingly safe creative pursuits, such as composition, could not manage to escape from the attraction of automation or machine-aid. Sound generation, by computer, allowed one to experiment with auditory perception, music grammars, and the development of

---

[32]   Binkley, T. E., op. cit., p. 9

[33]   LaRue, J. and Cobin, M. W., "The Ruge-Seignelay Catalogue: An Exercise in Automated Entries", Electronische Datenverarbeitung in Der Musikwissenschaft, Harald Heckmann, ed., Regensburg: Gustav Bosse Verlag, (1967), p. 41

original compositions. Computers had, by now, muscled themselves into such musicological areas as information retrieval, style analysis, the study of musical systems, and the development of music representations.

In the field of analysis, Arthur Mendel, Lewis Lockwood, Jan LaRue, and Harry Lincoln were interested in pattern recognition with a view to algebraizing characteristics of style for a particular corpus of music. Their corpora (the vocal works of J. S. Bach, the masses of Josquin, Haydn's symphonies, and the Frottola repertory respectively) were studied using 'overlay' procedures, where variant texts were compared for relevant similarities and differences—a technique not that far removed from linguistics.

Project suggestions and proposals soon began to emerge faster than the musicians could adopt and learn the relevant computing skills. Scholars took to suggesting that others should learn computer-aided research techniques, and that young students might benefit from a formal education in the area.

As well as 'overlay' expertise, the accruing and counting of statistical data was a favourite analytical technique employed by many musicologists. George W. Logemann, a computer scientist, studied specific problems via statistical analysis with the long-term goal of providing computational techniques which might be useful for the musicologist. In Logemann's eyes, it was interesting to count occurrences of certain features or combinations of features in compositions. Some combinations or features are possibly regarded as more important than others, leading to an analysis of the counts. As an exercise in statistical techniques, Logemann utilised the computer to find the entry point for the second voice in Bach's enigma canons from the "Musical Offering"—described in the conference reviewed by Allen Forte.[34] The theory

---

[34]   Forte, A., "Conference on the Use of Computers in Humanistic Reseach: A Review", Computers and the Humanities, 1 (1966-1967), pp. 110-112

was simple. Direct the computer to try all possible entry points and select those which lead to the best sound. However, the notion of 'sounding best' had to be expressed in computer terms. Normally, to answer complex questions requires complex processing, and the mechanisation of an intuitive process conjures up images of life-long calculations and over-sized machines. Logemann's computer program was surprisingly simple. The program obtained the distribution of intervals, and computed the total number of consonant, dissonant, and perfect intervals between the beat notes of the first and second voices (the test voice)—perfect intervals measured the 'openness' of the sound of the counterpoint. The difference between the number of consonant and dissonant intervals provided a quantity entitled 'harmonicity'. Something which sounds best (ie the 'correct' entry point for the second voice) should have maximum 'harmonicity'. To reduce the number of intervals possible, those intervals greater than an octave were decreased in size by an octave. Naturally, to obtain a more accurate 'harmonicity' value, different octaves should really have been taken into account. Thus, thirds and sixths were regarded as consonant; seconds, sevenths and tritones were regarded as dissonant; fourths, fifths and unisons as perfect intervals; and, harmonicity was calculated from consonance less dissonance. As might be expected, maximum consonance and minimum dissonance occurred for the same entry points for which harmonicity was maximised. An added advantage, and a notable one, was that the program and algorithms for solving Canon II (the test case) were equally effective in the solving of Canons I, III, IV and VI.

Logemann's research was very much geared toward providing tools for the musicologist and analyst. Described by Logemann as "the ultimate desire of musicologists",[35] and perhaps the ultimate task and challenge for Logemann,

---

[35] Logemann, G. W., "The Canons in the Musical Offering of J. S. Bach: An Example of Computational Musicology", Elektronische Datenverarbietung in Der

was the creation of "a complete library of all musical works and a programming system that will answer all possible questions".

As a natural consequence of working as curator of the New York Bartók Archives, Benjamin Suchoff undertook research into the feasibility of applying, via computer, Bartók's analytical methods to all the folk music Bartók had collected. The main task in hand was that of determining variant folk tunes or incipits. Determination of a variant may be likened to the determination of another incipit with high harmonicity or high similarity. Bartók defined the variants, to be determined, as melodies in which the pitch contour of various principal tones was entirely or partly similar.

A study which involves the examination of variant relationships calls for comparative analysis of thousands of folk melodies. Economy of means suggested the use of a computer. Suchoff opted to use DARMS for encoding the incipits, maintaining that "Ideally, one should transmute every music symbol into its equivalent graphic, then construct specifically delimited programs to process data in accordance with the special problem at hand".[36] Incipits arose from Bartók's own method of obtaining a skeleton form (ie stripped of ornamental tones) of melody sections which had different content structures. The width of a punched card determined the maximum length of an incipit, with longer incipits succumbing to truncation to fit the card.

Melodic, rhythmic, and melorhythmic variants (matching interval sequences and rhythm patterns) were deemed appropriate for treatment, and a

_____

Musikwissenschaft, Harald Heckmann, ed., Regensburg: Gustav Bosse Verlag, (1967), p. 79

[36]   Suchoff, B., "Computerized Folk Song Research and the Problem of Variants", Computers and the Humanities, 2 (1967-1968), pp. 155-158

FORTRAN program was used to process the incipits. Lincoln's approach entailed reading in each encoded incipit (a string of characters) and converting it into signed digits expressing the relationship of the intervallic succession. Converted strings were then stored in two disk files according to the polarity of the first digit (positive or negative). Sorting of each file so that positive preceded negative and less preceded greater occurred before final output of the files, with matching strings displayed in a single-space format, and the others in a double-space format (to emphasise identical incipits).

This method worked for melody but not for rhythm. Like most musicologists eager to use the computer, the more straightforward problems had been tackled first. However, an experimental program entitled "Bartók ARchives Z-symbol Rhythm EXtraction" (BARZREX), to convert rhythmic incipits into a lexically ordered sequence, was under way; and, melorhythmic variant determination (where strings of identical interval sequences were to be compared for rhythmic similarity) was on the drawing board.

Lincoln regarded the classification of melodies by interval sequence as a method superior to others. Intervals remain constant even when a melody is transposed, and thus provide an accurate picture of a melody's melodic contour. His system ignored repeated notes, but was still a major advance on the standard methods of alphabetising melodies—simply listing the letter names of notes. Lincoln, perhaps realising that more time spent in the initial stages of encoding melodic data would later permit a greater variety of operations, dropped any idea of an extended interval-encoding method in favour of DARMS for his research into the Frottola repertory. DARMS, intended to be used for encoding by non-musical clerical personnel, proved to be both economical and ingenious in the analysis and indexing of music.

The pilot study dealt specifically with sixteenth-century Italian vocal music (frottola)—a homogenous body of music which spans twenty-five years

25

(1505–1530) and contains about one thousand pieces, producing four thousand incipits. For the purpose of creating a computerised thematic catalogue, the incipit was fixed at seven interval changes, ignoring repeated notes. A retrieval program (written in assembly language and later to be converted to a more portable PL/1) would match an interval-sequence with its DARMS record and print it out in a more meaningful format. With an automated printing system promised for DARMS, printed musical output (in standard musical notation) was more than feasible.

Although the use of 'overlay' procedures was a technique not that far removed from linguistics, no major attempts had been made to apply other more important linguistic techniques to music. Terry Winograd, a graduate student in mathematics at the Massachusetts Institute of Technology, believed that such an undertaking was a profitable one, and discussed the possible use of linguistics for computer analysis of tonal harmony in his article "Linguistics and the Computer Analysis of Tonal Harmony".[37] Winograd was interested in expressing the specific structural and syntactic rules governing tonal harmony, in the form of a generative grammar. (In linguistics, a generative grammar is a set of rules whereby permissible sentences may be generated from elements of a language).

Applied to music, a grammar for harmonic structure may be broken down into five hierarchical-type elements. The first, composition, is a simple concatenation of tonalities, whilst tonality (the second element) contains the characteristic features of tonal harmony. The third and fourth elements refer to chords, with 'chord group' formed from local methods and 'chord' containing features such as root, type, inversion and linear function. Winograd

---

[37]   Winograd, T., "Linguistics and the Computer Analysis of Tonal Harmony", Journal of Music Theory, Vol. 12, No. 1 (1968), pp. 2-49

represented the last element ('note') numerically, considering traditional names merely as realisations. Even with the aforementioned five elements, tonal harmony is an extremely ambiguous language and provides many problems when converting its rules into a generative grammar. For example, an identical chord may appear at different places in a composition and have a totally different function in each place.

Using the grammar elements, Winograd created a program to give, for each chord of a composition "...its function within the tonality of which it is a constituent, its inversion, and the tonality hierarchy in which it operates".[38] The program accepted as its input a list of chords, each of which was a list of notes (where a note was designated by a note class and an octave). Written in compiled LISP[39], the program took about thirty seconds to process a Bach Chorale of about thirty-five chords. Schubert dances were also used as input, but the Bach was preferred since it offers a continuous sequence of harmonies and yet has a complex harmonic structure.

Simply offering analysis of harmony, and not all aspects of a composition's structure was the only drawback of the program. Winograd, however, claimed that the program (since it could decide whether a chord functioned structurally or linearly) gave sophisticated readings which were, more often than not identical to those of a human analyst. The distinction between structural and linear, however, is not always clear. In the tonal progression I-IV-V-I, IV could be regarded as linear or structural, and cannot be confirmed in either role if the composition is only analysed harmonically. The

---

[38]   Winograd, T., op. cit., p. 33

[39]   LISP (LISt Processing) is a high-level (ie uses English-like commands) computer
         programming language used primarily for list processing, symbol manipulation and
         recursive operations. A LISP program has the ability to modify itself as it is executing.

experiment demonstrated that it was possible to write a successful grammar for at least one aspect of tonal music.

Although, by the end of the 1960s, many musicological areas had harnessed computer power, the extensive field of music analysis still remained largely unexplored. Musicology was one of the last humanistic disciplines to become aware, and make use of twentieth-century technology. Only a handful of analytical programs were generally available, and no comprehensive theoretical system for computer-aided analysis existed. Any rapid evolution of analytical systems was often thwarted by the prospect of necessary tedious and time-consuming preparation, checking and correction of data. Most serious undertakings involve years of research, and the majority of researchers (when in the possesion of new technology) are always eager to progress as quickly as possible when competing for worthwhile results.

One of the more ambitious computer-aided analysis projects was that discussed by Eric Regener in "A Multiple-Pass Transcription and a System for Music Analysis by Computer".[40] The System for Analysis of Music (SAM) consisted of three main components. The first, a multiple-pass transcription code (LMT—Linear Music Transcription), could read through a score several times, collecting durations, pitches, text, literal information, and then alternate and variant readings—a different type of attribute on each pass through the score. The second, a two-stage assembler program written in FAP (an assembly language for the IBM 7090), comprised an input routine, abbreviation decoder, symbol recogniser and syntax checking routines in the primary stage; and, a table creator (to produce a table which held pitch-codes and durations etc, indexed by temporal position within the composition) formed the secondary

40   Regener, E., "A Multiple-Pass Transcription and a System for Music Analysis by Computer", Elektronische Datenverarbeitung in Der Musikwissenschaft, Harald Heckmann, ed., Regensburg: Gustav Bosse Verlag, (1967), pp. 89-102

stage. The final component, a retrieval routine, allowed access to any desired information by specifying up to four arguments (attack time, part number, voice number, and type of information desired). However, as the system was written in an assembly language unique to IBM machines, its transfer onto other computers to allow its use by other musicologists was not entirely feasible.

Bernstein's Sixteenth-Century Chanson research began as a pilot study into automated bibliography before it developed into the major project in stylistic analysis described by Marian Cobin.[41] Initially Bernstein, together with Joseph P. Olive, attempted to iron out the failings in the current manual methods of thematic cataloguing—failings such as the inability to discriminate accurately between two pieces which are essentially the same but begin with different material, and two pieces which are essentially different but begin with the same material.

Although their analysis programs enabled them to derive eleven types of stylistic data about a given composition selected from an initial repertory of three hundred pieces (which proved to be inadequate for testing their thematic cataloguing), they ultimately planned to encode the entire repertory of sixteenth-century chansons and extend their analysis programs to include even more parameters of style.

The eleven types of stylistic data fell into three categories. The first, 'Routine Analysis', involved the determination of the roots of all triads, measurement of the rate of harmonic rhythm, and determination of the range of vocal parts. 'Statistical Analysis' for each phrase included calculating the amount of relatively strong or weak root movement, the degree to which inversions were used, the ratio of complete to incomplete chords, and the

---

[41] Cobin, M. W., "Musicology and the Computer in New Orleans", Computers and the Humanities, 1 (1966-1967), pp. 131-133

frequency of recurrent harmonic adjacencies. The final category, a numerical representation of properties which defied verbal representation (entitled 'Analytical Observation') dealt with textual complexity and the interaction of several style parameters such as the number of voices, the amount of co-ordinated rhythm activity between voices, the number of separate rhythmic impacts across polyphony, and finally the durations of the notes themselves. The computer determined a numerical equivalent for each component of textual complexity. The numerical equivalents were then weighted according to their significance so as to produce an index of textual complexity for a given period of time.

Akin to Regener's system, all programs were written in FAP, and although fast (achieving an execution time of ten chansons every forty-eight seconds), the system was not portable. Their encoding language (CLML) however, made a bid for portability with its resemblance to DARMS, enabling entire chansons to be encoded part by part, leaving it to the computer to align separate voices.

Bernstein and Olive, unlike the majority of computer-innocent musicologists, at no time believed that the computer was in any way capable of analysing a piece of music. The fresh information made available by the computer was used purely as a means of substantiating insights already gained from human perusal of the scores. However, they did admit that since compilation of their statistical data would normally have taken a considerable length of time, it was not considered feasible without some form of mechanical aid.

Bernstein and Olive realised that the computer was not the answer to every problem. A computer can only resolve problems that are put to it in a precise and systematic form. However, if a problem is studied carefully and understood, then it may be expressed precisely and systematically by a

30

mathematical formula or language, and thus reproduced on an electronic computer. Early mechanical methods of composing music such as a toothbrush dipped in ink and sprayed across manuscript (where ink dots represent notes) and cards drawn from a pack to designate positions for barlines, pauses and rests, all failed because music is not simply a random selection of sounds but is instead, subject to its own unique laws and rules of construction. Musicologists were striving to understand music, express it via rules, mathematical formulae, and languages, with a view to either reproducing the music itself, or emulating the compositional process. Often the rules or formulae used for computerised composition were highly dubious and quite incomprehensible: "Briefly, the chord evaluation process is based on the computation of the ratio of the sum of the squares to the square of the sums of all the intervals in any given chord".[42]

Whilst the majority of computer-analytical studies dealt with music as notated, an important yet uninvestigated area was music as performed. Sadly, even with the creation of new areas for research, the contemporary attitude towards the use of computers was either unqualified acceptance or complete rejection. As far as some scholars were concerned, computers only had a future in incipit catalogues and concordances.

Arthur Mendel and Lewis Lockwood's interest in the possibilities of computer-assisted style analysis of Josquin Desprez's works led them to create an entire system based on Kassler's ideas and methods. Once a system has been designed and 'built', a music scholar does not have to concern himself with the internal workings of the computer—merely sit back and use the system. The 'black box' approach was adopted by Mendel and Lockwood, who

---

[42]   Hiller, L., "Some Compositional Techniques Involving the Use of Computers", Music by Computers, ed. H. von Foerster and James W. Beauchamp, New York: John Wiley and Sons, 1969

intended their system to be both usable and approachable by musicology students with no knowledge of computer programming.

Even with major upsets (the project's use of MIR for extracting information came to an abrupt halt when the university disposed of its IBM 7094 computer—all programs had to be completely redesigned for the new IBM 360 computer), the minor results achieved were claimed to be "of a nature that has not been achieved in any other use of the computer for style analysis in music".[43]

Although some scholars regard the character string as the most flexible and efficient format for representing musical scores, the proof-reading of strings involves so many factors that errors frequently creep in unnoticed, and the 'Josquin Project' was no exception to such encoding errors. Incorrectly encoded data produces deceptive and misleading results, and error-checking of data may be regarded as more crucial than the writing of analytical programs. Test programs for the Josquin data, however, managed to reveal a considerable amount of misprints, but results were nevertheless initially confined to the "Missa L'homme armé super voce musicales" since no one was confident that the rest of the data was error-free.

Some collating of statistics did eventually emerge. An early project obtained a list of locations and counts of all accidentals occurring in Book I and II of the Smijers edition—differentiating between those suggested by the editor and those reproduced from the original sources. Lockwood compiled lists of linear tritones and harmonic intervals involving accidentals, whilst Mendel noted the use of mensural signatures.

---

[43]   Mendel, A., "Some Preliminary Attempts at Computer-Assisted Style Analysis in Music", Computers and the Humanities, 4 (1969-70), p. 41

Since Lockwood and Mendel's analysis was essentially statistics collecting they were in danger of being flooded with a mound of data and no real idea of what it could be used for. Some results did confirm their suspicions relating to a suspect section ("Et in Spiritum") which contained a much 'lower-than-normal' proportion of incomplete triads. Of course, if the 'Et in Spiritum' section in other Josquin masses also contained a low proportion of incomplete triads, the claim that the suspect section was not by Josquin would lose its force. Thus, the only types of 'problems' tackled involved the counting and locating of specific features of the music.

Ever since the very early experiments in computer-aided musicology, music research had always been restricted by the hardware (machines) and software (computer programs) available—computers were built for science and not for the humanities. Converting mechanical methods of music composition into algorithms for the early computer often resulted in computer systems which were slower than the manual equivalent of quill on paper. Such disappointing results were no incentive for the conversion of those who believed that the compositional process could not be formalised—how can a mechanically-produced piece have the same creative power and depth of that nurtured and moulded by a human?

Time and time again, musicologists concluded that computers were only useful for statistical investigation and analysis of music theory (where analysis really meant collation of yet more statistics). Certainly, by the end of the seventies and the 'third generation' of computers (making advantage of integrated-circuit technology) computer-aided analysis results represented limited analytical procedures on limited data. Perhaps the many available theories of musical structure were so powerful and beyond explanatory scope that any form of computational formulation was inconceivable. Methods of computerised style analysis concentrated on comparatively superficial aspects

of music. So long as no claim was made to reveal 'the composing process' such a form of style analysis was acceptable. Obviously, since style must be reflected within the musical score, it is not unreasonable to assume that some method of computerised style identification is possible at a shallow level.

Musicologists were gradually making an effort to find out more about the mysteries of computers and modern-day technology. Although no straightforward text was available to introduce computing to the humanities' scholar, many researchers were pushing forward the idea of a 'music scientist'; "Most scholars who discover or can foresee a need for computer assistance in their work would do well, it seems to me, to learn as much as they can about the computer and, especially, about computing languages."[44]

Despite many doubts and fears, however, results from computer-aided musicology steadily appeared. Nancy Rubinstein succeeded in translating rules for 'franconian' rhythm (set out in 'Ars Cantus Mensurabilis' c.1260) into flowcharts and then FORTRAN.[45] James L. Curry identified the more common melodic contours used in a dissonant context within the Kyrie movements from five masses by Johannes Ockeghem,[46] whilst John Rothgeb's SNOBOL programs applied eighteenth-century rules to the harmonisation of unfigured basses—(as might be expected, the rules by themselves proved to be inadequate for the formulation of a general theory.)[47]

---

[44]   Kostka, S. M., "Recent Developments in Computer-Assisted Musical Scholarship", Computers and the Humanities, 6 (1971-1972), p. 15

[45]   Rubinstein, N., A FORTRAN Computer Program for Transcribing Franconian Rhythm, Ph.D. diss., Washington University, 1969

[46]   Curry, J. L., A Computer-Aided Analytical Study of Kyries in Selected Masses by Johannes Ockeghem, Ph.D. diss., University of Iowa, 1969

[47]   Rothgeb, J., Harmonizing the Unfigured Bass: A Computational Study, Ph.D. diss., Yale University, 1969

Naturally, results obtained from the many and varied projects ranged from failure, through inexplicable, to success. Don Cantor's and W. B. Barker's efforts to encode the two-dimensional aspect of music using a one-dimensional sequence of alphanumeric characters was, like the Josquin project, hampered by persistent errors (even though some scholars believed that music lent itself well to alphanumeric notation). They did, however, procure several computer programs which accepted and played back common musical notation, permitting audio-proof-checking of their data. Music could be input via a cathode-ray tube and stylus by manipulating music symbols on the screen.

Although computer power per square inch was on the increase, the price of current technology was still not low enough to attract the casual user— "Today it is possible for a person to buy an entire computer, complete with all kinds of fancy peripherals, for even less than most people spend for a new car."[48]

As a demonstration of how computers could be harnessed for music analysis, Ian Bent and John Morehen addressed themselves to the question, "How could we write an analysis of the form of a piece of music using the computer?"[49] By way of an introduction to computers and analysis, Bent and Morehen described three of their computer programs. The first program used a 'parsing' approach devised by Allen Forte, where a string of musical code is repeatedly rewritten in a simpler form until there remains no repetition of musical material, whereupon the composition can be expressed symbolically in terms of letters, ie AB for binary form and ABA for ternary.

---

[48]   Howe, H. S. Jr., Perspectives of New Music, Princeton University Press, Vol. 16, Fall-Winter 1977, p. 71

[49]   Bent, I. and Morehen, J., Proceedings of the Royal Musical Association, 104 (1977-1978), p. 32

The second program tackled the hitherto ignored subject of textual underlay. Since the carefully documented rules for text placement in sixteenth-century polyphony were concerned with the rhythmic characteristics of individual voice parts, and most computer input codes encoded a voice at a time, the rules lent themselves easily to translation into a simple numeric coding system. Within the coding system utilised, a five digit code could represent a note's pitch, duration, position within ligature, and whether or not the note carried a syllable. The underlay program aimed to detect passages of polyphony which contravened the strict rules of sixteenth-century textual underlay. A third program, also concerned with textual underlay, searched a composition for a specified combination of rhythm and syllable placement.

In addition to the three programs, another suite of programs which Bent and Morehen described, set out to allocate syllables to textless music. The programs, so the authors claimed, could provide a "definitive, or nearly definitive answer" and were a step toward showing that the computer could be of considerable assistance in a hitherto uncharted area of research.

Some of the Bent and Morehen programs used rules to allocate syllables to textless music. The use of rules to recreate or create music was nothing new, since both Maurita and Ronald Brender were interested in describing traditional music using rules (a formal grammar), and Terry Winograd was intrigued by the possibility of a generative grammar for tonal harmony. This sixties' research was nothing to the scope of that of Mario Baroni and Carlo Jacoboni.[50]

Baroni and Jacoboni set themselves the task of generating a grammar which described the Bach Chorales. The grammar itself was to be used to

---

[50]   Baroni, M. and Jacoboni, C., Proposal for a Grammar of Melody: The Bach Chorales, (Montreal, 1978)

generate 'correct' musical phrases by computer. Unlike a certain DEC 11/70 that was fed all the most-used words in every Western movie in a bid to persuade the computer to write a Western story, Baroni and Jacoboni claimed that the analysis of their material was not statistical. Gilbert Bohunslav's DEC 11/70 produced a Wild West yarn which made no sense at all, whereas Baroni's and Jacoboni's analysis of a 'cropped' set of phrases (60 versions of 36 phrase pairs) resulted in a set of rules which produced 'correct' Bach-like phrases. The rules of the grammar, and a carefully selected set of generated melodies made publication. Although Baroni and Jacoboni claimed that statistical analysis was of no benefit to those who try to discover something about the grammatical structure of the musical language, statistics formed the basis for the rhythms used during the 'casual' generation of the phrases.

Many scholars still had doubts regarding use of statistics for analysis. Mark Ellis, however, hoped that an increased use of statistics would aid progress in the study of a composer's mind. Ellis attempted to move away from the current 'structural' and 'style from statistics' methods of analysis and, noting that fugue subjects in J. S. Bach's "Well-Tempered Clavier" book one had more repeated notes than those in book two, attempted a computerised quantative study of the fugues. The computer was adopted to increase efficiency—ie reduce time and human error. Ellis side-stepped the 'are statistics a waste of time?' question by suggesting that if results were of interest to an analyst, the analysis could be considered valid. 'Facts' have to be related back to the score before they can become informative and thus useful.

Leo J. Plenkers used a statistical method to research into a possible correspondence between two different thirteenth-century repertories. Linguistic pattern-matching software, created by the Computer Department of the Faculty of Letters of the University of Amsterdam, was adopted for the project. The software, effectively an early 'query system', allowed interactive searching and

the use of a 'don't care' mechanism with the search criteria—ie look for B flat, A, C, 'something'. The analysis, therefore, was a manual analysis, and the computer was merely an aid to provide the statistics for analysis.

"...have you ever heard the saying: 'He uses statistics like a drunken man uses lamp-posts—for support rather than illumination.'"[51] John Morehen outlined a selection of statistical tests to which he believed Renaissance polyphony was particularly well suited.[52] Such tests involved an examination of the difference between bass parts of four-part and five-part pieces, determination of chronology, and determination as to whether an instrumental piece might have originated as a vocal piece. The results, never entirely conclusive, were still helpful enough to encourage him to pursue statistical analysis further. The statistics, however, were used to confirm previously conceived questions and notions, rather than to provide fresh insight and illumination. Is not the aim of analysis to illuminate?

John Morehen's results were encouraging, but not all computerised analyses were successful. "It seems that my dream of a facile computer-assisted analysis of the structure of the three similarity relations is turning into a disaster."[53] Running John Rahn's similarity function (ATMEMB) on a VAX 11/780 required almost one hundred hours of solid computer time. Dealing with pitch-class sets, the ATMEMB function was intended to return values between zero (minimum similarity) and one (maximum similarity).

---

51   Computing, 10 November 1988, Backbytes, p. 128

52   Morehen, J., "Statistics in the Analysis of Musical Style", Proceedings of the Second Symposium on Computers and Musicology, Orsay, 1981, (Paris, CNRS, 1983), pp. 169-183

53   Rahn, J., "Toward a Theory for Chord Progressions", Proceedings of the Second Symposium on Computers and Musicology, Orsay, 1981, (Paris, CNRS, 1983), p. 83

Although the computer was not always successful at analysis, it never failed to impress as a time saver. Computer aid was justifiable when used for confirmation of a theory, as opposed to 'push-button' 'black box' analysis. The majority of analysis was in fact a reworking of material, providing new insight and stimulating new directions of thought. All this, with technology built for science and not humanity. Music was, not surprisingly, one of the last disciplines to make use of such technology.

The goal for many was a complete library of musical works stored on computer and a programming system which could answer all possible questions—ie discover what makes music catchy and appealing. The main techniques used by music scholars involved statistical interrogation and feature counting, probability, use of grammars for computerised composition as an analysis check, set theory, information retrieval, and style analysis; but, no matter how outwardly artistic, all these techniques relied upon some form of mathematical analysis. All techniques were based, effectively, upon the detection of repetition and similarity—although many scholars had no confidence in the computer's ability to detect similarity.

The early Eighties saw an increase in the availability of both general-purpose software and 'accessible' hardware. The 'personal' computer enabled many to carry out their own research at home and on an individual basis. Prior to that, lack of suitable software tools was a major obstacle to fruitful research. Each researcher had to start from scratch, inventing computational methods before any analysis could take place.

Although musicologists recognise that computers can be beneficial, many lack the technical knowledge required to harness the computer's power. Some are unaware of the time and effort necessary to achieve their objectives. General-purpose application software helps, but blinkers the computer into answering questions which are often tangential to the musicologist's original

queries. A special musician's toolkit, a software package offering many tools to perform small tasks—tools that can be bolted together to perform larger tasks—will offer the musicologists versatile software that will remove the blinkers and allow even the less technical to achieve results.

Analytical Methods

The "Concise Oxford Dictionary" defines the verb 'analyse' as "examine minutely the constitution of". "Roget's Thesaurus" lists three words analogous to the verb 'analyse'. The first two, 'class' and 'inquire', are the terms most often associated with analysis. When a market researcher stops a passer-by in the street and asks a series of product-related questions, the researcher is merely inquiring and searching for information with a view to classifying the passer-by among his or her already-acquired statistics, ready for yet more analysis. The third term listed in the Thesaurus comes closer to the true definition of analysis—'to decompose', to separate into its simpler elements or constituents. In the "New Grove", Ian Bent applies the "Concise Oxford Dictionary" definition of analysis to music, and defines musical analysis as "the resolution of a musical structure into relatively simpler constituent elements, and the investigation of the functions of those elements within the structure".[54] He later writes that "Analysis is the means of answering directly the question 'How does it work?'".[55] Analysis, therefore, is any close study of the score (or sound produced by performance of the score) which helps the researcher to understand what makes the composition 'successful', ie 'work', and extends the answer to Pinkerton's question, "What is it about simple melodies that makes them so widely appealing?".[56]

There are many documented techniques for finding out how compositions 'work'. These 'analytical methods' are quite varied and diverse in nature, but all help a researcher in his understanding of a composition's

---

[54] Bent, I., "Analysis", New Grove Dictionary of Music and Musicians, ed. Stanley Sadie (1980)

[55] Bent, I., ibid.

[56] Pinkerton, R. C., "Information Theory and Melody", Scientific American, 194 (Feb., 1956) pp. 77-86

structure. Most of these methods are manual (ie performed by hand) and require a thorough knowledge of the analytical technique. An expert in one particular method is not necessarily an expert in any other method. If these methods are implemented on a computer system, a researcher will not require an in-depth knowledge of the analytical method to achieve results.

'Fast analyses', analogous to 'fast food', will hit the market place. Phrases such as "quick Schenker" or "double Reti and a dash of Ruwet" could become as common as "double cheesburger and large fries". The 'burger bar' staff procure results by assembling a few ingredients. The computer analyst extracts information by assembling a few tools. Musicians, academics, and interested parties, with only a limited knowledge of the required methodologies, can generate analyses for their own perusal. Scholars should not, however, dispense with learning new analytical methodologies. The better the understanding a scholar has of a particular technique, the greater the benefit the scholar will receive from examining any joint venture analyses—ie those produced by computer and human.

One cannot assume a cumulative benefit from applying different analytical methodologies to a composition. Whipped cream is not normally associated with hamburgers, but the customer always has the option. The benefit of an Analysis Environment which contains many tools to emulate either complete analytical methodologies or parts of methodologies—tools which can be combined in many ways—is one of experimentation. With a large collection of tools, an analyst will be tempted to explore the possibilities of combining the more obscure and interesting of them. It might be fascinating to see what further analysis can be applied to the results of a previous analysis.

It is not too much to expect a musicologist to learn another analytical technique. A musicologist, however, might simply wish to see what sort of results a particular technique will produce, before committing himself to learning

the new method. One might argue that the musicologist can examine manual results of the analytical technique, but the computer can be used to apply the technique immediately to compositions of interest where manual results are not readily available.

Results from a Schenkerian analysis, even if produced by a computer, will still require a basic knowledge of the Schenkerian technique to interpret the results. To carry out an analysis by hand requires an in depth knowledge of the methodology. Tedious and repetitive operations must to be performed by hand, and are prone to error. Why not give these tasks to the computer? Why not share the analysis with the computer? The musicologist and the computer should be a team. A simple awareness of the method will allow a musicologist to interpret the results without having to undertake the whole analysis.

A single tool does not have to perform a whole analysis. There may be tools however, which when combined, execute a complete analysis. Using current computer technology, it is easy to make a quick change to a computer program before trying it out again. Although it is not deemed good practice in the computer programming world, the Analysis Environment is trying to encourage experimentation. Bolt a series of tools together to see what information is produced. The information is interesting. Make a few changes to the usage of the tools, add an extra tool here, take out a tool there, and try it again. The fresh information is often even more interesting. Even if the information is unhelpful, it takes only a moment to rethink the construction strategy and try an alternative method—to experiment.

Researchers will not be limited to one particular method. If all methods are available on computer, the researcher can select from a variety of methods, and is able to combine results to provide a more thorough analysis. At present, it is difficult to combine the results of one method with that of another. If all methods are available on a computer system, a procedure can be fashioned for

43

'bolting' the methods and results together. Before such a computer system can be outlined, the available analytical methods must be evaluated, and the more suitable set aside for computer-implementation.

"You get an idea; at some point another idea kicks in; you make a connection or a series of them between ideas; a few characters (little more than shadows at first) suggest themselves; a possible ending occurs to the writer's mind..."[57]

Man has inspiration, an idea. From that and other ideas, he creates a complete work of art via an elaborate and highly complex process which no-one totally comprehends. The work of art might well be based upon a single idea— perhaps a subconscious idea. The subconscious idea rests at the back of the mind in a solitary form. However, the moment it becomes outwardly apparent, it changes and alters itself, and is no longer the sole idea, but a host of transformations and variations. The transformations and variations find a suitable medium, and they merge into a final form—the work of art. Such an opinion of composition is not reserved solely for the author of this thesis. Zaripov stated that "Music occurs in the consciousness of the composer 'in a burst of inspiration', subconsciously, intuitively...".[58]

Rudolph Reti regarded music as a linear process which passed from a 'beginning' to an 'end', rather like a chain. The chain, containing links which overlap and occur side by side, evolves from a motif inside the composer's mind which he allows to grow through constant transformation. If left untouched, a single inceptive subconscious idea continues to bud, and produces multiple transformations of itself, concatenating into a never-ending

---

[57]   King, S., The Stand, Hodder and Stoughton Ltd., 1990, p. 10

[58]   Zaripov, R., "Cybernetics and Music", Perspectives of New Music, 7 (1969), p. 117

chain. It is only through the will of a composer that this reaction is cut short and aborted. Composers often talk of the difficulties involved in 'finishing' a piece. Although a fade-out executes a specific gestural intention, it is not unknown for tape compositions to resort to a fade-out as an ending in a bid to combat either incessant inspiration or perhaps simply the lack of a studio technique. "Even Mozart apparently found it easier to begin a movement than to end it: think of all the works left incomplete".[59] Ideas grow. How can the swelling beast be terminated? Premature and prolonged endings are a common cause of compositions which 'sound wrong'. Endings, nailed onto pieces out of sheer terror of uncontrollable unceasing thoughts and images, always leave the listener in the cold. Thus the composition of an ending is probably more demanding than any other section of a piece.

Schoenberg believed that the very first notes written by a composer should be taken seriously by the theorists, whilst Heinrich Schenker's influence made many theorists take little interest in the last notes of a composition.[60] Esther Cavett-Dunsby likened a piece, and its ending, to that of a detective story or a 'Whodunnit?'.[61] The real enjoyment of a story is its end. Who committed the crime? How incomplete a story would be without its end. The ending of a composition can be likened to the last chapter of a book. The length of the final chapter is dependent upon the length of the book. Lengthy books require lengthy final chapters. Shorter, less-complex books require little in the way of a conclusion and usually warrant a shorter final chapter. Large and complex compositions are very rarely brought to an end in one bar. The

[59] Cavett-Dunsby, E., "Mozart's Codas", Music Analysis, 7:1, 1988, p. 47

[60] "With the arrival of the 1 [last note of the Ursatz—see page 48] the work is at an end. Whatever follows this can only be a reinforcement of the close—a coda—no matter what its extent or purpose may be". Schenker, H., Free Composition, trans. and ed. Ernst Oster (New York: Longman), 1979, p. 129

[61] Cavett-Dunsby, E., "Mozart's Codas", Music Analysis, 7:1, 1988

"Haffner" symphony (K385) of Mozart, for example, reiterates the last chord eleven times before the double bar is reached as if to say "Yes, this is the end. This is the end. This is definitely the end". Even so, a listener unfamiliar with a composition does not know whether its ending has started or is only part of the way through. A 'new' listener can only say "that was the ending" during the moment of silence between fall of baton and auditorium applause. The applause itself is usually led by those who know the work. "Has it finished?" is a question frequently asked at musical premières where the composer and performers are initially the only ones familiar with the composition. Every note of a composition is, therefore, one number of a vast combination which will unlock the door of understanding. If a composer spends precious time and thought in the placing of every little ink dot onto manuscript, all of a composition is important, and as such, an analysis in pursuit of more understanding and insight into a piece, might be more successful if it examines all notes.

All too often one cannot appreciate the creative skill and technique behind a work of art. One longs to know how such a large and complex work can possibly evolve from apparently nothing. A search for the creative processes which went into its construction, and quantification of those processes in such a way that other budding musicians might benefit, will procure some initial ideas or 'building blocks' from which the entire art form has evolved. Schoenberg stated that "A composer does not, of course, add bit by bit, as a child does in building with wooden blocks. He conceives an entire composition as a spontaneous vision".[62] The dramatists would have us believe that many prodigious composers conceived their compositions as 'spontaneous visions'. Unless a composer has reached the stage whereby he can imagine, in a moment, the complete and final version of his musical work, and then retain it

---

[62]   Schoenberg, A., Fundamentals of Musical Composition, ed. Gerald Strang, Leonard
         Stein, London: Faber, 1970, p. 1

in its entirety in memory until such a time whereby he can write it down, composition is as the term suggests, a building up from small pieces, stage by stage. Those with the gift of spontaneous vision warrant the title prophet or seer, not composer. Yet, there are many stories of composers completing major musical works on the eve of their performance. Such feats as this can only be possible if the total structure and layout of the work is held in memory.

Conceiving an entire piece as a 'spontaneous vision' sounds very impressive, but it contradicts the fact that music unfolds over time. The vision is more likely to comprise an initial harmonic or melodic idea, a starting point, an end point, and a climax. What happens between start, climax and end is most likely to evolve within the composer's mind over a period of time. Writers, for example, very rarely conceive entire novels in an instance. They have an idea, perhaps a storyline and the bones of structure. The meat develops over time. Numerous novelists, for example, begin writing with no clear idea of where their story is going, or how it is to end. The story develops into a trilogy or series. Stephen King began writing a series called "The Dark Tower" in 1970. The first volume, "The Gunslinger", in a series of unknown length was published in 1982 and King wrote, "At the speed which the work entire has progressed so far, I would have to live approximately 300 years to complete the tale of the Dark Tower."[63] King conceived the tale in 1970, but his vision was not one that embodied the whole tale. In the first volume he found himself writing about characters and events for which he had no explanations. "But what of the gunslinger's murky past? God, I know so little. The revolution that topples the gunslinger's 'world of light'? I don't know. Roland's final confrontation with Marten, who seduces his mother and kills his father? Don't know. The death of Roland's compatriots, Cuthbert and Jamie, or his adventures during the years

---

[63]   King, S., <u>The Dark Tower. Volume 1: The Gunslinger</u>, Sphere Books Ltd, 1989, p243

between his coming of age and his first appearance to us in the desert? I don't know that either. And there's the girl, Susan. Who is she? Don't know."[64] To date, the tale is still incomplete, and volume three has just been published in an estimated series of seven. At the end of the third volume King writes, "The course of the next volume is still murky."[65]

Prodigies aside, composition has to be learnt using building blocks in the manner of a child learning co-ordination and structure with its building blocks. As such, for the majority of composers who never reach the rank of 'seer', musical compositions have to be built up using standard building blocks. Schoenberg realised that students of composition would not initially be able to envisage the 'spontaneous vision' and attempted to teach the "Fundamentals of Composition" via examples of musical building blocks and how, through the ages, composers have linked together and varied such blocks to create large-scale compositions. In music analysis, similar building blocks can be linked together to create large-scale analyses.

Schenker regarded all tonal compositions as a 'projection' in time of the tonic triad. The tonic triad is projected by its transformation into an 'Ursatz' and the 'prolongation' of the Ursatz. The Ursatz itself, comprises a melodic linear-descent to the root of the triad with a bass progression from tonic to dominant and back to the tonic. Prolongation involves ornamentation by auxiliary, passing, and scalic notes etc of the Ursatz. Even Schoenberg suggested that all tonal compositions resembled a tonic-dominant-tonic cadential progression—"In a general way every piece of music resembles a cadence, of which each phrase will be a more or less elaborate part. In simple cases a mere interchange of I-V-I, if not contradicted by controversial harmonies, can

---

[64] King, S., ibid., p248–249

[65] King, S., _The Dark Tower. Volume 3: The Waste Lands_, Sphere Books Ltd, 1992, p512

express a tonality".[66] To suggest that all composers of tonal works have the tonic triad at the forefront of their mind before and during composition is a rather strong generalisation. Certainly, employing Schenkerian analytical techniques, it is possible to progress backwards from a finished work to achieve the goal of a Schenker 'Ursatz'. Students, when battling with the concepts of Schenkerian analysis, frequently refer to the technique as that of reducing a masterpiece to "Three Blind Mice". The 'Ursatz', in effect a harmonised descending scale to the tonic, usually from the third (hence "Three Blind Mice"), together with various variations is shown to be the basic structure behind the creation of many tonal works. However, interesting though this theory may at first appear, in reality it implies that all tonal compositions are not only based upon the same fundamental structure, but employ similar compositional techniques, ie elaboration and variation. Schenker was not wrong to declare that all compositions may have an underlying structure or idea. The idea is more likely to be an idea which remains unique to each composition, and not one which is common to all. A 'variable' Ursatz is a more viable concept—an Ursatz which obeys certain structural rules, but looks different for each composition. An automated tool for locating such Ursatz-variations would help to ascertain if Schenker's theories extend beyond the realms of tonality. After all, the songs of Franz Schubert are outwardly tonal, but many of them cadence in a different key from that which they start in. No standard Schenker 'I-V-I' Ursatz could form the backbone of these pieces. Experiments carried out by Nicholas Cook[67] suggest that listeners only have a direct perception of tonal closure—a movement or work beginning and ending in the same key—when the time scale involved is in the order of a minute or less.

---

[66] Schoenberg, A., op. cit., p. 17

[67] Cook, N., "Music Theory and 'Good Comparison': A Viennese Perspective", Journal of Music Theory, 33.1, 1989

Standard underlying 'Ursatz-type' progressions should not be ignored, however, since the searching process itself will procure further information on structure, and aid serious study of the composition. Certainly, a mechanised-utility which could suggest the possible location of a standard Ursatz within and underneath a composition will save time and energy, and allow the scholar to concentrate on other areas of interest. The very attempt to discover how an art form is created from specific melodic and harmonic ideas will give a 'fresh' insight into many of those masterful creative processes which for so long appear to have remained a mystery. The aim of analysis is after all, the answer to questions and not the art of creating analytical methodologies—"...you only have to look through today's specialist analytical journals to realise what a high premium is generally put on the formulation of increasingly precise and sophisticated analytical methods more or less as an aim in itself".[68]

Music scores for works such as the "Sonatas and Interludes for Prepared Piano"[69] by John Cage, employ standard Western musical notation. However, any melodic analysis via the score, of such a composition, will yield false and misleading information. What the classically-trained musician sees and imagines in the score will bear little melodic resemblance to what he or she hears when the piece is performed on the 'prepared' piano. The "Sonatas and Interludes" are playable, and at times melodic pieces when rendered on a standard unaltered piano. The sounds which emerge from such a performance, however, bear little resemblance to the composer's intentions. Cage imagined something rather more rhythmical and percussive. Any true performance of these pieces requires the piano to be 'prepared' with various objects and substances such as screws, bolts, nuts, rubber and plastic inserted between

---

[68]   Cook, N., A Guide to Musical Analysis, 1987, J. M. Dent and Sons Ltd., p. 3

[69]   Cage, J., Sonatas and Interludes for Prepared Piano, 1948

the strings at carefully measured distances along the strings. An analysis of the score will yield structural information, but cannot begin to convey the actual sound. The score has become a stepping stone from imagination to reality. Cage delighted himself in writing compositions which were designed to confuse and unsteady the listener. The "Sonatas and Interludes" have this air of 'trickery' where sounds heard during performance are not backed up by the written score. The only feature which remains consistent between sound and score is rhythm. If one ignores the pitches, reads the score as though entirely written on a monotone, and uses a phonetic-type sound for each note, the character and mood of the piece is brought across better than through a rendition of the score on a normal unaltered piano. The "Sonatas and Interludes" of Cage were written primarily for their rhythmical qualities (an attribute not overly obvious from a simple glance at the score) and Cage, intending to evoke the same sounds as the Gamelan, turned the composition of "Sonatas and Interludes" into an exercise in rhythm.

Sadly though, there are far fewer methods of rhythm analysis, than there are of melody analysis. Many musicologists argue that melody is 'more obvious' than rhythm, and the majority of musicologists, when analysing compositions, opt for melody analysis rather than rhythm analysis. Further suggested methodologies for rhythm analysis would be welcome, and to be able to select from a variety of mechanised tools to aid in such analyses would be even more welcome. A single methodology or a single tool is not enough to tackle every type of composition type, but a plurality of methodologies in itself does not provide answers to questions. A language dictionary which contains the meaning of only one word offers little help when translating a passage of text. A dictionary containing sixty-thousand words is much more useful but cannot be used to provide a meaningful translation without a basic knowledge of the language and its grammar. Likewise, a selection of methodologies enables an analyst to select the method best suited to the task in hand, but the method

itself does not provide answers, only a means of helping the scholar formulate his or her own answers.

An analysis, to be truly effective, should have its results portrayed in the same medium as that of the work under analysis. Thus, an analysis of a score should yield a score. Many methods of music analysis do indeed produce scores, with the reductive method of Schenker being perhaps the most obvious, creating not a true music score, but a graph or over-view of the whole work instead. However, Schenker's graph concept is not entirely successful because it employs rhythmic symbols to indicate levels of significance within the graph, and as such, is only readable by those with a true knowledge of the Schenker symbols. Music analysis and actual analyses should be accessible by anyone interested and not simply those who have been grounded in the academics of music, or indeed those who have been grounded in Schenkerian techniques.

If the only way to enjoy a story was to read a book (which requires a knowledge of the alphabet, grammar etc) and not to listen to a recording or watch a film, some people would remain unaware of stories and the benefits and morals that they can portray. The old adage that a contingent of monkeys typing for an infinite period will eventually produce the complete works of Shakespeare can be applied to music analysis. If we can undertake an analysis with little or no experience, many results will be trivial, but some might be enlightening and beneficial. A fresh approach, an approach untainted by years of academic study could produce new information, or show current information in a new light. Music analysis should be accessible by all.

Keller said that "Music about music is immeasurably more objective than words about music, because music is absolutely concrete". Keller's method of presenting a music analysis involved composing a score, using the same instrumentation of the original scrutinised score, which contained passages of the original score interspersed with aural demonstrations of the links between

them. This meant that anyone could sit back and simply listen to an analysis without requiring an academic background in music. However, his musical analyses were actually based upon the written score, and since they were not in fact music (sound) about music (sound), but were music about scores, he was not quite practising what he preached.

Keller's scores portray his analyses of music scores. Anyone with the ability to read a music score can read Keller's scores and probably understand his analyses. When the scores are performed, however, they become subject to further analysis by the performers, the 'score about score' becomes 'sound about score', and the final analyses are those of the performers and not Keller.

If one can read and comprehend words, an analysis written in words, about words, will also be readable and most likely comprehensible. If one can read a score, an analysis depicted as a score is again readable. Likewise, an analysis portrayed in sound is accessible to all who can hear.

In listening live, however, one cannot dip randomly in an out of sequence as in reading. With a book or score, one can turn back to a passage to read it again and make comparison, which makes the score or text a more suitable medium for showing the results of analysis.

Music about music, words about words, and scores about scores will work, but an analysis intermingling them all allows a greater variety of people to benefit. Those who are unable to read music can read the textual notes or even listen to analytical extracts.

Semiology, the 'science of signs', when applied to music proceeds in two stages. Firstly, the musical structure is broken down into distinct units. This segmentation process produces the musical equivalent of 'signs'. Secondly, the usage of the units in relation to each other is examined. Most forms of semiotic analysis retain standard music notation, and rearrange the original score, bar

by bar, section by section, into columns of matching or similar paradigms. Semiology breaks a music structure down into small units which become nonsensical if broken down further—a unit of three pitches which recurs within the composition is likely to be more significant than a unit containing a single pitch which recurs within the composition. However small or large the unit, it must be salient or motivic to have semiotic significance. The relationships between the units are examined and set down using standard music notation. The units as entities in their own right are not significant, but considered within the network of relationships which constitute the musical structure, they become significant. A scholar examining a semiotic analysis can immediately see the relationships between musical ideas and where certain ideas originated within the score.

In an effort to create the sounds which can only be heard inside the mind of a composer, many contemporary works break away from the confines of standard music notation. Sounds heard whilst at a tender age, now stored within our subconscious, creep into the very notes we inscribe. "he [the composer] is not always aware... of how the melody was born and took shape in his consciousness".[70] Maxwell Davies, although referring to the actual score as much as the aural content, wrote with reference to his "Symphony", "When I started the present work, in 1973, I had no idea that it would grow into a symphony".[71] Do composers themselves know what they really want? Do they know what they wish to achieve? They, and their minds are corrupted by the sounds which surround them. Over the centuries, composers have been accused of plagiarism. They write what they perhaps imagine to be original music, only to discover at a later date that their supposed 'new' melodies have

---

[70]   Zaripov, R., ibid.

[71]   Maxwell Davies, P., "Maxwell Davies: Symphony", (notes on the back of a record sleeve), DECCA 1979, HEAD 21

been inspired, albeit subconsciously, by the works of someone else, and yet, the melodies are probably not the rightful property of that composer either. No doubt, the melodies were ingrained into their minds when they heard them some years prior to their regurgitation.

Analysing actual sound, even with the technology of today, is by no means a simple task. Computers can be set up to store, in real time, sound data of live and recorded performances. Nicholas Cook, for example, developed a computer program for the analysis of piano performance.[72] The input for the analysis is an ordinary audio recording of the music, whereas the output from the analysis is a listing of the times and intensities of attacks in the performance. Since the output from the analysis is a list of figures, the output may be subject to further data processing by the computer. The input to the computer is in effect an interpretation or analysis of the score by the pianist, and makes the performance itself available for objective analysis. The data accumulated within the computer during a recording is not normally of any immediate use to a musician. The data is usually a large series of complex numbers, sometimes represented by a graph, and needs further processing to provide an analysis of style and structure. Converting seemingly endless lists of digits and aesthetically pleasing graphs into a medium which displays the structure and style of the composition in a clear and legible way is more often than not impractical.

The 'medium about medium' theory suggests that the analysis of sound should yield sound, and certainly, being able to sit down and listen to an analysis is a most attractive proposition. In the same way that background music can be appreciated without the need for intense concentration, a

---

[72] Cook, N., "Structure and Performance in Bach's C Major Prelude (WTC I): An Empirical Study", Cambridge University Music Analysis Conference 1986

'background analysis' might also inform the listener, again, without the need for intense concentration. After all, music is an experience over time, and a sound analysis producing sound would reflect this. There are, however, a multitude of recorded performances available for analysis. All performances are different, and a performance is an expansion upon the score, in effect an analysis in its own right, and perhaps the last step of composer-intended elaboration. Many composers imagine the sound and not the score. If one was to compose using a reverse Schenker technique (which is hardly standard composing practice) an Ursatz would become sound via a series of elaborations which progressed through background, middleground and foreground levels, and finally through the score and then the performance. To analyse sound, really to analyse an analysis, would only explain the analysis and not the musical score or indeed the music itself.

When learning a composition, ready for performance, a musician must interpret and analyse the score. Some scores are very detailed, but the majority are vague and ambiguous when stating how certain phrases should be played and what significance they have within the composition as a whole. Music notation is an incomplete indication of performance. The musician must make his or her own judgements and analyse the score. All performances are different, but the score remains the same. The analysis of a score (a fixed and constant medium encompassing the thoughts of a composer) is the best approach available. The score is a stepping stone from imagination to reality. Interpretation of this stepping stone is the art of the classical musician.

In fact, if all performances are different, one might argue that a single performance should be heard once only, and allowed to be absorbed into the mind. Mistakes (all parts of a performance) gradually disappear, and the whole merges into an overall atmosphere which is often the only real representation of the original thoughts of a composer. In a musical performance the audience

lapses into a state of semi-consciousness. The notes disappear into the overall sound, shape and form. A 'wrong' note or an action which does not fit the form, draws their attention and upon waking from their stupor, the 'wrongs' are actually the things they remember.

Scores can be analysed in rigorous and abstract terms. Scores contain pitches and time-points, but listeners hear tunes and harmonies and not the notes as distinct separate entities. "Music need not be performed any more than books need to be read aloud, for its logic is perfectly represented on the printed page; and the performer, for all his intolerable arrogance, is totally unnecessary except as his interpretations make the music understandable to an audience unfortunate enough not to be able to read."[73] This implies the death of music recordings. Every performance is different, however, and it is the arbitrary elements in the sounds we hear which make music so wonderfully attractive. To hear repeated recordings of one performance, infuses our minds with an interpretation or analysis which is not rightfully ours. Examining the score, rather like reading a book, forces a reader to imagine his or her own story or performance. Watching the film of a book often destroys the magical images conjured up during its reading and frequently forces the film director's own interpretation and analysis upon the viewer. Likewise, listening to the performance of a score often destroys the magical images conjured up by its reading and forces the conductor's interpretation or analysis upon the listener.

In the same way that performances of a composition are quite different, analyses of a composition are also quite different. There is no 'correct' performance. There is also no 'correct' analysis, and each analysis emphasises different points and conveys different information to the reader. A computer

---

73   Newlin, D., <u>Schoenberg Remembered: Diaries and Recollections (1938–76)</u> (New York: Pendragon Press, 1980), p.164

then, to be of ultimate use to the musician, should not impose a definitive set of results. The computer should work hand in hand with the musician to arrive at a set of results via a regular and consistent input from the musician. That way, the individuality of manual analyses will still come across in those produced with computer aid.

Analysis, of any kind, involves repetitive and laborious procedures such as deletion of repeated phrases or sections, matching of similar patterns, reduction of elaborate material, possible shifting and repositioning of particular phrases, and identification of potential transformation and variation between phrases. These are all types of procedure which, at first glance, appear readily adaptable for computer. If the act of analysis is made easier, more scholars who would previously have stayed clear of the many seemingly complex methodologies will be tempted to 'test the water' and try out the easier alternatives. With a plethora of easy-to-use automated tools available, it becomes difficult to resist the urge of experimentation—if only to see what it can achieve. The results produced will generate further interest and experimentation.

No manual analysis methodology can possibly ascertain the significance of every note within a composition during the lifetime of the analyst. Such a task is not of finite complexity. A computer-aided analysis, however, will ensure a more complete and thorough examination of all notes. If one does desire to create a new analytical method, the analytical method must have been carefully defined and tested manually beforehand in order to achieve a smooth and pain-free transition from theory to computer implementation. Constructing a model from plans which are not logical will procure a model which fails to work. If the manual methodology does not perform correctly, a computer-aided version will not perform correctly either. Computers may be fast, but they are not magicians. If a manual methodology has not be tested, computer limitations will

twist and warp the initial analysis objectives and methodology during their implementation.

Computers, 'number crunchers', may only seem good for analysing music in a logical and mathematical manner. Some theorists argue that music is mathematical, and even the general public have a tendency to link the two subjects together. If music is not mathematical, and is deemed to be purely intuitive, mathematical and logical computers will require skilful programming if they are ever to reproduce the intuition of a human. H. F. Cohen stated that "In his [Kepler's] view, God, in creating the Universe, was guided by certain mathematical regularities. Hence for man, the greatest insight into nature that can possibly be gained is the discovery of these same regularities as they are expressed in the world". Many compositions are indeed composed in a mathematical way, using certain scales, sets of notes, or a serialistic twelve-pitch mechanism for selecting and combining notes. Computer-aided analysis of pieces composed via these mathematical methods are often more successful than similar analyses of pieces composed via more intuitive methods, but in the case of computer-aided analysis it is usually the analytical method itself which comes under criticism. Any method of analysis designed specifically for a computer, creates the added danger of 'cold' and often machine-like results, which do not reflect the constructive art behind the composition. "Set Theory", originally developed by Allen Forte for computer analysis of works composed by way of a serial technique, functions admirably. Serialism, a technique which ensures that melodies contain a single occurrence of every semitone of the chromatic scale, has to be mathematical in its approach. An analysis, however, which yields a result such as: "Every note occurring in the piece is contained within the set '10-5'", (where the set '10-5' contains ten semitones of the chromatic scale, and does not account for different octaves) raises both questions and eyebrows, and can be likened to describing a book as employing a set called 'alpha' where the set 'alpha' contains every letter of the alphabet.

Statistical analysis, described by many as 'feature counting' since it involves recording the number of occurrences of a specific feature, is also a methodical and mathematical form of 'analysis' which adapts effortlessly to computer. Computers are excellent at counting quickly, and thus, after a score has been converted into a form readable by computer (ie employing an alphanumeric encoding language to represent pitches and durations as letters and numbers—of which a multitude exist) it is possible to produce any statistics required. "How often does an F sharp occur in the piece?", "What is the range of the tenor part?", "How frequently does the composer employ a diminished second?", are all questions easily answered through the writing of small computer programs or algorithms. From statistical analysis, all one can retrieve is information such as : "Piece A is longer than piece B", "Piece A employs more semiquavers than piece B", "Piece B contains three E double-flats", and the usefulness of such results is questionable taken out of context and used as an entity in its own right.

For a computerised analysis to be successful, the analytical method should not be one designed specifically for a computer, but must instead be an adaptation of some already-working manual-methods. The analytical methods must be tried and tested before the computer implementation begins. Thus, if it were possible, an error-free computer implementation of the Schenker method (a method tried, tested and documented) would be far more useful than any purely mathematical method simply dreamt up for the computer.

Many pitfalls must be overcome when endeavouring to implement analytical methods on computer. The first, and perhaps the most daunting, is the transfer of a score into a medium which can be stored within a computer. This entails converting all the wonders of the full-score into a code which consists of simply alphanumeric characters. In a way, even the conversion of a score into representative letters and numbers involves some analytical

reductive processes. What should be included and what should be omitted? Many coding systems do exist, and most, in an attempt to avoid the 'analysis-during-encoding' trap[74], like the notorious DARMS—"Digital-Alternate Representation of Musical Scores"[75] try desperately to encapsulate every symbol which might appear in a score. Consequently, the amount of stored data becomes vast, and highly complicated. Coding systems can be simplified, but in such a situation, supposed non-essential markings in the score (slurs and phrases for example) have to be left out. Thus, the simpler and easier a coding system is to use, the less useful the encoded data becomes. Encoding language design should take into account ease of use and comprehensibility, offering a fine balance between the two.

A normal score itself, is a poor representation of the sounds which a composer imagines, and therefore, an encoded version of the same score becomes an even worse representation. As a result, since a computer analysis of encoded data produces encoded data, any output from the analysis, to be of any use, has to be converted back into, or related to normal music notation. There can be no doubt that essential information is lost at both ends of the score-to-code and code-to-score processes.

There is still a great deal of research to be undertaken in the quest for the ultimate encoding language—'standards' groups still meet in an effort to

---

[74]   Encoding every symbol and mark on a score would take up a great deal of time, effort and computer storage. In an effort to reduce any or all of these factors, analysts preparing scores for computer-aided analysis become selective in the items they encode. This very selection process involves an analytical procedure to determine what should and should not be left out. What one scholar dismisses as immaterial might be regarded as significant to another. This subjective process is known as the 'analysis-during-encoding' trap.

[75]   See chapter three, starting on page 72, for a description of the DARMS encoding language.

standardise on the many languages and systems available[76], and, until a complete system emerges, a truly complete analysis will not be possible. The thoroughness of a computer-aided score analysis can only be guaranteed if the encoded data totally reflects the score. However, it is not unknown for researchers to spend so long creating a method of converting a music score into letters and numbers, that no time has been left for the actual analysis. For the purposes of this research, two encoding languages have been used: a canonical version of DARMS, and a language designed by Walter Hewlett.[77]

From the computer implementation of analytical method emerges the dream of a 'push-button analysis'. The dream stars a music scholar pushing a key on a computer keyboard to produce complete analyses utilising the techniques outlined by Schenker, Reti, Ruwet, and any other musicologist he cares to select. There is, however, no set answer to be obtained from a particular analytical technique, and even the apparently specific rules of Schenkerian analysis provide controversial results, with, for example, frequent arguments as to whether or not the "Ursatz" behind a work is an "eight-one" or a "five-one", ie descends from the tonic to the tonic, or from the dominant to the tonic. Analysis though, rather than providing 'the' answer (or even 'an' answer) should only attempt to give fresh insight into a composition—to generate new ideas or even extract an "I never knew that" response from a user. Any new knowledge about the structure, meaning, or purpose of a composition (no matter how small), should be a major goal of all analyses.

---

[76] Musical Interchange Processing Standards (MIPS), for example, is a subcommittee of the American National Standards Institute (official name ANSI X3V1.8M) and is dedicated to developing an encoding language which can express scores written in standard music notation.

[77] See chapter three, starting on page 72, for a description of the Hewlett encoding language.

Music scholars can roughly be divided into two categories: those familiar with analysis methodologies, and those unfamiliar with analysis methodologies. For those who are familiar, a 'push-button' system is a hindrance, not an asset. Such scholars do not wish to be restricted by someone else's computer programs. Their manual skills, acquired over the years, can be adapted to suit any situation. Pushing a single button will only invoke the computer's sole means of applying the analysis methodology. For those unfamiliar with all the intricacies of a particular methodology, however, a 'push-button' analysis is a boon. Despite being unfamiliar with a specific methodology, they can press a single button to obtain results from any selected methodology. At the very least, the results acquired will generate further interest in either the methodology itself or the actual score. In order to achieve the perfect 'push-button analysis', one which omits nothing, a computer should scan and examine every note contained within a composition. Every note has a relationship with the inceptive ideas of the composer. Despite this, the majority of analytical methods deal only with monophonic data on the grounds that a complete monophonic line throughout a composition conveys a composer's style—"...the essence of polyphonic style must be embodied in the manner in which a composer constructs an individual line.... Consequently, the style of a composer must be enshrined on the printed page in a single monophonic line, although that line would need to be of sizeable duration (eg a complete Mass movement or motet section) for meaningful deductions to be valid"[78]. Monophonic analysis leads to melodic analysis because a monophonic line usually contains melody. Even transferring melodic analysis onto computer has its problems, since the computer needs to know just what a melody is. Defining melody. however, is not easy, and certainly to convert any human definition into a computer

---

[78]   Morehen, J., "Statistics in the Analysis of Musical Style", Proceedings of the Second International Symposium on Computers and Musicology, Orsay, 1981, (Paris, CRNS, 1983), p171

algorithm is a tremendously complex task. The four definitions offered in the "Oxford Concise Dictionary" ("Sweet music; musical arrangement of words; arrangement of single notes in musically expressive succession; and the principal part in harmonized music") offer little help in the quantification of the word "melody", and, without quantification, one can only suggest that a computer has to analyse every possible melodic route through a composition to be certain of missing nothing. More often than not, the highest sequence of pitches in a composition constitutes the melody, but as this is not always true, it cannot be used as the basis for finding melodies. Many composers hide contrapuntal melodies within the underlying harmony of a composition, making it wise to examine every possible melodic path through a composition in order to guarantee the 'capture' of the melodies. Composers frequently let melodies weave a path through underlying harmony—melodies which would normally be missed if one was to regard only the highest notes of the composition as melody. Melody could be regarded as the 'focal line'. A set of preference rules could be used to decide what the focal line is. For example, by default the top line could be regarded as melody. If more frantic movement occurred in another line, that line becomes the melody. If another line is louder, that line becomes the melody, and so on.

If the composition, following successful encoding, is held in a two-dimensional array or grid, with sounding notes placed in the grid corresponding to where they are situated in the score, it is a relatively simple task to write an algorithm facilitating the calculation by computer of every route through the grid, and thus every possible melodic line of the composition. Simple maybe, but without an algorithm to remove superfluous melodies, the computer produces a seemingly mammoth amount of permutations and output. For a hypothetical composition of ten chords in length, with each chord containing three notes, the number of possible forward melodic routes is fifty-nine-thousand and forty-nine—a veritable wad of paper to sift through.

With the length of the average composition greater than ten chords, one might think that thorough searching of the grid is a fruitless task, and indeed, until computer technology for the individual advances in terms of searching and pattern-matching algorithms, processor speed, memory and disk capacity, it also appears to be an impossible one. Currently, the majority of computer analyses deal only with single-line encoded melodies. Likewise, although the research outlined in this thesis concentrates primarily on melodic analysis, it also describes some tools for basic harmonic and rhythmic analysis. Since there is little limitation in the expansion capabilities of the proposed Analysis Environment, the addition of other tools for contrapuntal and more advanced harmonic analysis is quite viable.

In general, most analyses concentrate on pitch, since pitch, although not always the case, is often regarded as being of more importance than rhythm. In piano performance, for example, subtle changes in rhythm will frequently go unnoticed (and are often taken for granted—labelled as 'rubato' or 'performer's licence'), whereas changes in pitch are rather more obvious and are usually branded as 'wrong notes'. "When music is played, the durations of notes are slightly altered, notes emphasised and envelopes constantly changed in order to increase the emotional effect. It has traditionally been the job of the music's interpreters—musicians and conductors—to add this emotional human touch to the mainly mathematical directions of a score"[79]. On the other hand, the opening bars of Beethoven's fifth symphony would lose all significance and meaning if the first three notes were given different durations.

If an analysis provides new information, it has achieved its goal. The encoding language is of little importance in a computer-assisted analysis when all that is required is some form of result. Admittedly, if everyone utilised the

---

[79]   "Putting Passion into Micro Music", The Australian, 9 September 1986

same mechanism for converting scores into letters and numbers, scholars could exchange encoded data with each other. However, at the end of the day, obtaining a 'result' is more important than the means whereby the result is obtained. Likewise, the analytical method used is largely irrelevant if the method enlightens the scholar. The environment outlined and proposed in this thesis refrains from imposing a specific technique upon the user, and instead provides many ways of combining different analytical methodologies, offering many routes for expansion.

Music composition can be assimilated to Meccano. An examination of the basic 'Music-Meccano' set reveals the nuts and bolts of composition—pitches and durations—and, without these, it becomes difficult for a music structure to exist. Creating a composition without pitches and durations is difficult, but not impossible. In fact a handful of composers spend most of their time attempting to create music structures without the aid of standard nuts and bolts such as pitches and durations. Further inside the set lie the standard hinge, bracket, and coupling pieces, analogous to the set patterns of scales and arpeggios. Harmony, regarded by many as the meat and foundation of composition, has its equivalent in the strength-giving girders and plates of standard Meccano. Shape-defining parts, such as funnels, wheels, and hooks, may be likened to melody. Musical compositions, therefore, may be fashioned through assemblage of Music-Meccano parts. Even standard parts though, can be assembled in an unorthodox fashion to create an abstract form, revealing a structure which does not merit the title of model or composition.

The last item within the Meccano set, and possibly of more importance than any other, is the instruction manual. Certainly, without the manual, beginners would find it difficult to build anything of significance. Surprisingly, nowhere within the glossy leaves of such a manual are there explicit details of how to build a specific model. Instead, alongside a picture of the completed

model—containing most of the information necessary for building—are enlarged and cut-away illustrations of the more difficult to see, and comprehend sections. From these illustrated sections it is possible to realise the underlying structure of the model, and progress backwards to discover not only which parts are employed, but also how they are employed. Thus, through a reductive and cut-away process, the Meccano manual succeeds in revealing both utilised constituent parts, and methods for transferring such parts into finished models.

It follows that a Music-Meccano manual, rather than contain pictures of mechanical objects, should contain pictures of musical objects, ie scores. A Meccano manual might contain models such as a Lorry-mounted Crane, Steam Press, or Swing Bridge; A Music-Meccano manual, however, might contain scores of a Tenor Song, Piano Sonata, or Clarinet Solo, together with cut-away and reduced score-illustrations showing the techniques used to build the scores from the Music-Meccano parts.

The research outlined in this thesis attempts to create a Music-Meccano 'set', containing girders, plates and the nuts and bolts to connect them together. The Music-Meccano set (referred to as the Analysis Environment) will allow analytical methodologies to be constructed by bolting specific tools[80] together using a technique called 'piping'.[81] Rather like real Meccano, which has sets ranging in size from a 'pocket set' to the grandest of grand 'set ten' (housed in a wooden chest of drawers and costing more than the average adult can afford, let alone a child), the size of the Analysis Environment is dependent upon the

---

[80] A tool may be defined as a computer program designed to do a specific task. The output of the tool may be altered through the use of options. A tool is very efficient because it is designed to do only one task.

[81] Piping is a mechanism used to connect two tools together by using the output of one tool as the input to another tool.

number of tools within the environment. To use MSDOS[82] terminology, all the tools are 'external' to the environment—ie they are individual programs which reside on the hard disk of the computer rather than in the memory of the computer. The proposed Analysis Environment can be likened to a 'set one' Meccano set where a limited number of parts (tools) allow many different models to be made. Major models cannot be made without a larger number of parts and, likewise, the Analysis Environment will need to be expanded—a not altogether difficult task—via the addition of new and extra tools in order for it to allow the building of larger analytical methods from within it.

"There are a large number of analytical methods, and at first sight they seem very different; but most of them, in fact, ask the same sort of questions. They ask whether it is possible to chop up a piece of music into a series of more-or-less independent sections. They ask how components of the music relate to each other, and which relationships are more important than others. More specifically, they ask how far these components derive their effect from the context they are in".[83] All analysis theories, no matter how seemingly diverse and apparently incomprehensible, generate new ideas and show a composition in a different light. All of them, therefore, are worthwhile and should be readily available as usable tools in a computerised environment. However, the tools incorporated into the 'set one' Analysis Environment have had to be limited in order to provide a finite length to the research period. Goals of the research have expanded on numerous occasions, always when the previous collection of goals were nearing completion. With even a modest amount of tools, there is always the danger of imagining that the Analysis Environment will be able to answer every question. In 1987, a circular from "Art-Science

---

82   Microsoft Disk Operating System—MSDOS is a series of commands which enable a
      user to manipulate data on an IBM or compatible personal computer.

83   Cook, N., A Guide to Musical Analysis, 1987, J. M. Dent and Sons Ltd., p. 2

Workstations of New England" announced the completion of the Music Scholar's Workstation (MSW) - "a most comprehensive and flexible music and sound research tool"; "... Providing unparallelled power, speed and user interfaces within a software/hardware system which has virtually unlimited flexibility, the Music Scholar's Workstation embodies the finest enabling technology for exploring the relationship between theoretical constructs and psychological reality". Art-Science Workstations fell into the trap, and the circular was marketing 'licence' as opposed to fact. Those with a true knowledge of the facts, described it as "...slicing, dicing, midi, ai, kitchen sink etc".[84] No product can provide 'the' answer for everyone. One scholar's goals when analysing music will be entirely different from another scholar's goals. "The decision as to which music properties are 'interesting' and worthy of observation is a subjective one. This can, and should, change depending on the goals, experience, and means of the investigator".[85] A scholar can, however, insert his or her own tool into the Analysis Environment in order to extract an answer for the particular problem at hand. Provided that the tool has been added using the standards set out in chapter four of this thesis, any future users of the Analysis Environment will be able to utilise the new tool.

The primary task, before experimentation with the Analysis Environment, is to determine what tools should be present in 'set one'. Reti regarded music as a chain. A basic tool to locate and suggest the constituent parts of a chain should be in 'set one'. Likewise, the endings of musical pieces, since they are often the most difficult and lengthy phase of composition, are very important. Examining the ending of a composition in particular might provide important

---

[84]   Hawley, M., "Arts and Sciences Workstations of New England", Music-Research Digest, Vol. 2, Issue 7

[85]   Brolsma, B., "Music Analysis Presentation Formats", Music-Research Digest, 1987, Vol. 2, Issue 6

information. An 'ending' tool should also be included in 'set one' (the Analysis Environment).

Schoenberg simplified the compositional process by dividing compositions up into building blocks. Tools to help locate the building blocks and discover how they are put together should also be present. Locating the Ursatz behind a composition—'the' building block—could be a primary task of another tool in the environment. Locating the Ursatz of a composition is often enlightening. The Ursatz, however, should not be regarded as specific locations of I and V chords (which is a widespread but not very useful literal interpretation of Schenker graphs), but as the longest prolongation of I-V-I, ie three time spans. The Analysis Environment should contain a mixture of tools for applying reductive processes to music. Altering the method in which the tools are bolted together will change the degree of 'reductiveness'. The tools may be used to search for an Ursatz, but they may equally be utilised for some other purpose. Although Schenker's Ursatz concept is restricted to tonal composition, an adaptation of the tool for locating a standard Ursatz could be used for locating a 'user-defined' 'Ursatz'.

The tools suggested so far are all score-based. Whilst tools for actual analysis of sound are outside the scope of this research they could be included subsequently in 'set two'. However, the inclusion of a tool for 'performing' results from score-based analyses will allow scholars to 'listen' to their analyses—akin to Keller's ideology. Rhythm, like sound, is not given the attention it deserves from the analysts. In any well developed analysis system, tools for rhythm analysis must be included or easily added. Several rhythm analysis tools, some of which use a pattern-matching technique, have been incorporated into the Analysis Environment.

Semiotic or 'layout' analytical tools, which provide a more visual form of analysis, have been incorporated. A tool for printing melodies, extracted from

the score during the execution of an analytical tool, is also in the set. The printing tool does not provide output of a publishable standard, but its output is quite readable and uses music-like notation.

Forte's comprehensive 'Set Theory' (a ready-made mathematical method for analysing atonal music) warrants a tool in the Analysis Environment. However, since Forte spent a great deal of time writing computer programs to utilise his methodologies, there seems little point in reinventing the wheel. Similarly, statistical analysis converts to computer algorithms with relative ease. However, many computer-aided research projects fall into the statistics trap where all computer programs are merely 'feature counters'. Only one tool, therefore, will be purely statistics based.

Each tool will be created to perform a specific task. Options may be specified when using the tool, and these options will only change the style of output from the tool. Since a tool will only be designed to perform a single task, it will perform it efficiently. More complicated tasks can be accomplished by bolting a series of tools together.

The thirty-two tools comprising the 'set one' Analysis Environment can be combined in almost four-hundred different ways, providing many methods of extracting information from the written score.

The Encoding Languages

"Only someone who has prepared programs and data for computer processing can fully realise how misleading in its suggestion of ease is the phrase '...and then he fed it to the computer'."[86] In the 1987 "Directory of Computer Assisted Research in Musicology"[87] a special section on music encoding languages cited seventeen examples of encoding methods in use at the time. What this dearth of languages shows is that despite the great number of 'standards' committees meeting to decide what should and should not be used for scholarly research, most scholars are inclined to create their own encoding language to suit their current needs rather than learn and use an already established system. "A lot of wasted effort and, perhaps, heated controversy could be avoided by seeking agreement among researchers in music on a standard computer representation for music that will serve all of their various needs."[88]

Some encoding languages treat only one attribute, whereas others cater for scores of any level of complexity. The DARMS[89] encoding language was conceived for use with a photon printer (an early typesetting device) and as such had a symbol or set of symbols to encode absolutely everything in a musical score—from text to slurs. "The most advanced output method of all, the Photon printer for which DARMS was developed, is today seldom mentioned in

86   Lincoln, H., The Computer and Music, Cornell University Press, Ithaca and London, 1970, preface page xi

87   Directory of Computer Assisted Research in Musicology, 1987, Centre for Computer Assisted Research in Humanities, Menlo Park, California

88   Slawson, W., "A Book Review: Computer Applications in Music", Journal of Music Theory, Vol. 12. No. 1, 1968, p. 108

89   Erickson, R. F., "DARMS, A Reference Manual", (New York: Queens College, CUNY, 1976)

a musical context, and it has evidently passed quietly away."[90] The photon printer never emerged, but the encoding language is still in use today. Music scholars saw that DARMS could be put to uses other than music printing. It offers a method for encoding anything musical a scholar so desires. It does, however, become more and more complex to use when the amount of information to be encoded increases. The unabridged or "canonical" version of the language is vast (the documentation was available a few years ago, to those with the relevant underground contacts, in a huge A4 binder) and consequently there are many possible condensations of the language. Most scholars, who recognise the importance of standards for music data exchange, resort to DARMS and use a subset to encode only the items of interest to them. Much of the data encoded in early projects was tailored to answering specific questions. There were many subsets of DARMS in use. Since scholars only coded the items of interest to them, there became a need for a 'canonizer' that would expand these subsets into a complete and full DARMS representation of the score.

When time is limited, adopting an already standard encoding language allows research to begin almost immediately. A friend and scholar began work on the analysis of an Elgar symphony. Time was limited, but he decided that to create an encoding language was the first step necessary to fruitful research. Several months later, and at the end of the total period allotted for all the research, the design of the encoding language was almost complete. The actual analysis never took place. The infamous Josquin project used teams of people to encode Josquin's Masses. The project, however, never really got off the ground and the data—never proved to be totally error-free—remains as a memorial to yet another ambitious project which never was. "...the tape

---

90   Kostka, S., "Recent Developments in Computer-Assisted Musical Scholarship", Computers and the Humanities, 6 (1971-72), p. 17

containing the Masses is I hope somewhere in a locked cage in the basement of Firestone Library at Princeton, or, somewhere in the Woolworth Center (the music building) at Princeton."[91] Such is the desire for ready-coded data, many scholars still have a hankering for the non-standard base of data locked away in the vaults of Princeton. There is a vast quantity of encoded musical data available, but because of the lack of a single widely adopted encoding method, the quantity of consistent data available for serious research is scarce.

Theoretically, to be sure that all information possible can be gleaned from an encoded version of a score, every item on the score must have a coded equivalent and actually be encoded into the data. Too much information is always better than too little information in the first instance. It is easier to create a short tool to extract the relevant data required for the current research. To this end, the *darmstrip* tool (described in chapter four and outlined in appendix A) does just that—it extracts only the data required, for the Analysis Environment, from canonical DARMS data files. Obtaining data files containing canonical DARMS is a problem though. When one is coding a score for oneself, the only way to save time—although false economy—is to skip on the items selected for coding from the full score. Canonical DARMS data files are few and far between. Those who produce them are perhaps either doing so for the sheer love of collecting encoded data, or because they have been tasked to do so by some higher authority. Music archives do exist, although the majority of items stored within their vaults are printed music scores and not the required tapes or disks containing data files of encoded scores. The Oxford Text Archive[92] has now diversified and is accepting encoded versions of musical works. No attempt has been made to standardise on the method used

91   Earp, L., "Princeton Josquin Data", Music-Research Digest, 1987, Vol. 2, Issue 2

92   Oxford Text Archive, 13 Banbury Road, Oxford OX2 6NN, England

to encode the scores, but at least a ready base of musical data is beginning to evolve.

It is self-evident that before encoding of a score begins, archives should be checked to see if such data already exist in an encoded form. By the late 1960s, central banks of data were beginning to emerge. Most of the data stored in these new and accessible data banks, however, was restricted to incipit or index data. "A start in the direction of a central bank has been made in Binghamton, where the author's indices of the frottole, parts of the Italian madrigal repertory, and the complete works of Palestrina have been joined with other researchers in sixteenth-century music."[93] Despite the amount of data filtering into the Binghamton data bank, most of it was not full scores, ie all information on the score.

While over in England for a round-table discussion on computers and music, Walter Hewlett presented the Oxford Text Archive with the complete Preludes and Fugues of J. S. Bach in encoded form. Since Bach's original manuscripts had very little in the way of dynamics and other markings, Hewlett's encoding of the preludes and fugues (containing essentially just pitch and rhythm) can be described as comprehensive. His data, in fact, formed the basis for the analysis and test of the Analysis Environment, outlined in chapter five. The encoding method used for the Hewlett data, however, is neither DARMS, nor well known.

If data do not exist for the score under analysis, an attempt should be made to encode the score using a standard and established method. Some designers of encoding languages claimed that after half an hour of practice, the symbols and characters of the encoding language could be typed faster than

---

93   Lincoln, H., "The Thematic Index: A Computer Application to Musicology", Computers and the Humanities, 2 (1967-68), p.220

writing the music out by hand—"The point is made that after half an hour's practice, the researcher has learned the code so well that its symbols can be typed faster than writing out the neumes by hand"[94]. Certainly, DARMS was designed to be used initially by non-musicians to speed the input of data, and as such, can be a very fast and efficient encoding method to use. Despite its distance from musical notation, many musicologists soon became fluent in DARMS in the same way that language students become fluent in French or perhaps German. A good encoding language has to strike a balance between readability and ease of use.

Alphanumerics (letters and numbers) are as similar to music as chalk is to cheese. Even so, most musicologists saw that the 'text string' (a sequence of letters and numbers) was the best way to represent music inside a computer. "Music lends itself extremely well to being handled as a string. ...the string can represent as much or as little of the music as one wants..."[95] Some encoding methods involved only numbers which made computer-processing more straightforward, but encoding and data checking much harder.

Hundreds of encoding methods exist. Each new computer-aided music-analysis project seems to generate a new encoding language, and the bulk of them escape documentation. The majority of methods reflect the requirements of their creators and are used to represent the information required for specific research projects. Some describe musical contour, but ignore rhythm. Others symbolise pitch, but ignore octave position. DARMS, however, has the ability to describe everything. "...the great encoding scheme controversy evidently has

---

94   Bowles, E., "Musicology and Computers", Computers and the Humanities, 4 (1969-70) pp. 207-219

95   Bent, I. and Morehen, J., Proceedings of the Royal Musical Association, 104 (1977-1978), p. 32

subsided into a rather placid acceptance of DARMS."[96] Interestingly enough, the Hewlett encoding method was not one of the seventeen listed in the "Directory of Computer Assisted Research". A questionnaire sent out by the Directory had ninety-nine responses describing projects involving the encoding of musical information. Only twenty-nine of the projects used well established encoding methods such as DARMS. Those familiar with the problems of music encoding are not convinced that a standard for music encoding is a viable goal. The path toward this goal is worthwhile, if only for the accessible data and information it will provide.

Since DARMS is destined by many to be the encoding language of the future, selection of data encoded using the Hewlett method might seem unwise. The Hewlett data, however, exists. It may be encoded using an unusual method, but it does exist. If the Analysis Environment tools operate on DARMS data, the Hewlett data can be converted (via the *htod* tool described in chapter four) to DARMS notation.[97] Using a ready-encoded set of data enables the analysis and testing to take place almost immediately, rather than waiting for the encoding to end, the error-checking to finish, and the debugging to cease. The data used for testing purposes is largely irrelevant since it is the concept of joining small tools (which perform small tasks) together, and using them to help answer complex analytical questions, that is of greater importance and requires testing. The Hewlett data also saves time and effort in other areas. The question "What is a part?" has already been answered because the Hewlett data is encoded in parts. The subjective decision has already been made as to what is and what is not a self-contained part. The fugues themselves are relatively straightforward to sub-divide into individual melodic lines or parts.

---

[96]   Kostka, S., op. cit., p. 20

[97]   The essential elements of both the DARMS and Hewlett encoding methods are described later in this chapter.

Other musical works, and in fact the end of some of the fugues, are rather more problematical when it comes to distinguishing parts. "Polyphonic music is represented in tracks [parts]. Each track is supposed to represent a single voice. For vocal and most instrumental parts, the representation process is straight-forward. But for some instrumental music, especially certain lute and keyboard pieces, the decomposition of the musical fabric into separate tracks is often arbitrary and problematic. Where extra notes occur simultaneously, we have tried to avoid adding extra tracks by allowing a single track to split into chords. All notes of the same chord must be of the same length, otherwise they will be put onto separate tracks. Arpeggios and other free-style figuration present especially difficult problems for this kind of representation."[98]

The Hewlett data is encoded one part at a time. Each part is delimited by the words 'BEGIN' and 'END', and all the parts of a single fugue constitute a single data file (diagram A).

Diagram A.

original
Hewlett
data-file

BEGIN

encoding of part 1

END
BEGIN

encoding of part 2

END
BEGIN

encoding of part n

END

---
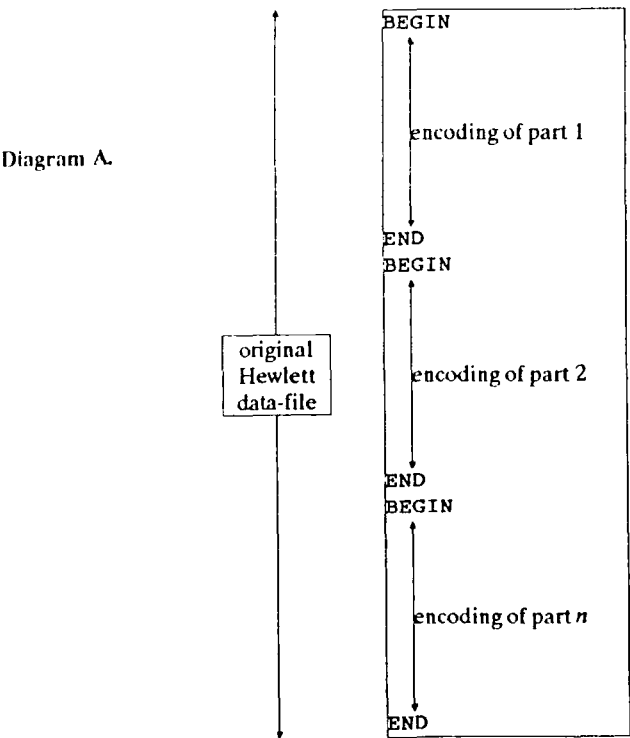
[98]   An extract from the leaflet which came with the Hewlett data.

78

The parts must be separated into individual data files for use in the Analysis Environment. A four-part fugue, for example, will have four associated data files within the Analysis Environment (diagram B).

Diagram B.

```
          BEGIN                BEGIN     BEGIN     BEGIN
                               |         |         |
           encoding of part 1  part 1    part 2    part n
                               |         |         |
          END                  END       END       END
          BEGIN
                             ( new data-) ( new data-) ( new data-)
                             ( file 1   ) ( file 2   ) ( file n   )
original   encoding of part 2
Hewlett
data-file
          END
          BEGIN

           encoding of part n


          END
```
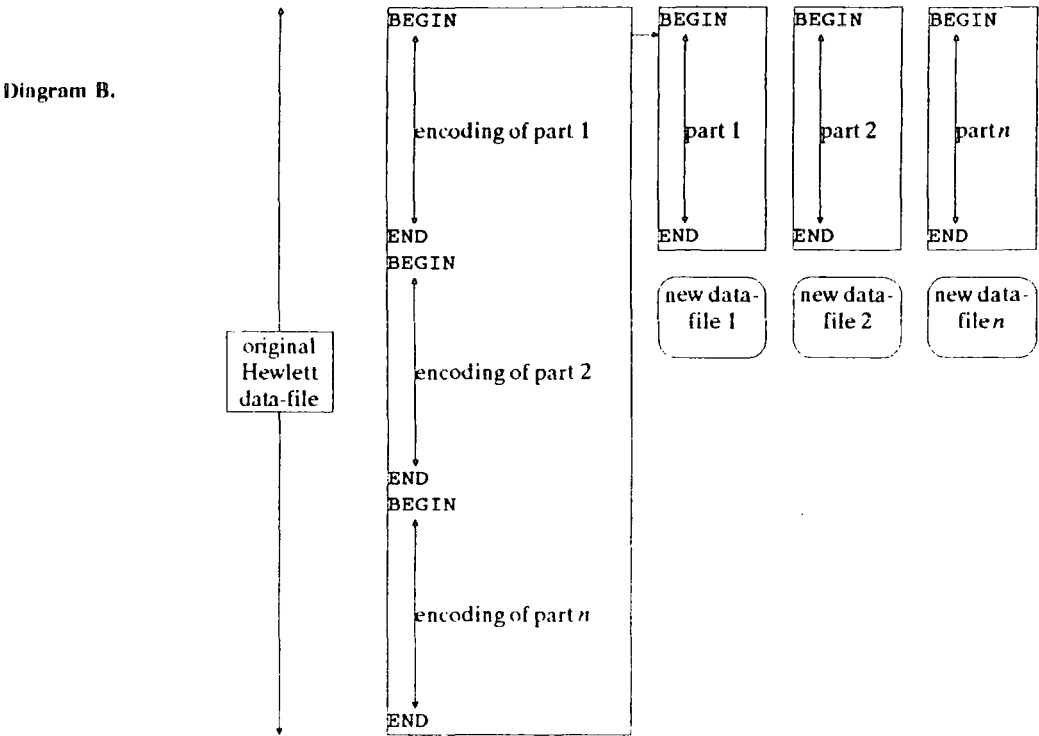
Notes are placed on separate lines which aids readability and takes up nominally more space than squeezing as much information as possible onto a single line. Bars are preceded by the word 'measure' and the number of the bar (diagram C).

Diagram C.

```
                    measure 1
                    note
                    note
                    note
                    note
                    note
                    note
                    note
                    note
                    measure 2
                    note
                    note
                    note
          data-file note
                    note
                    note
                    note
                    note
                    measure 3
                    note
                    note
                    note
                    note

                    etc.
```
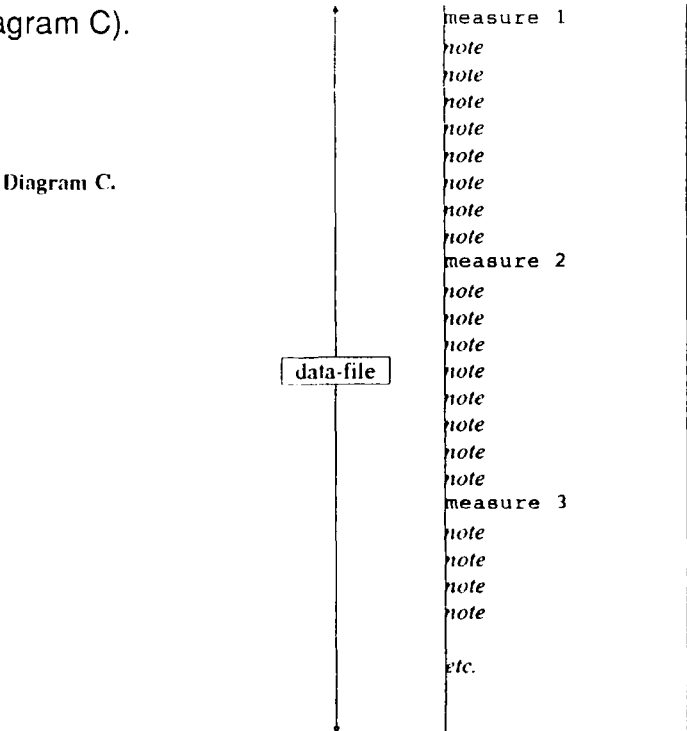
The notes themselves consist of a pitch code, a sharp or flat sign, an octave position indicator and a duration code (tied notes are encoded as a single pitch with a single duration) (diagram D).

Diagram D.
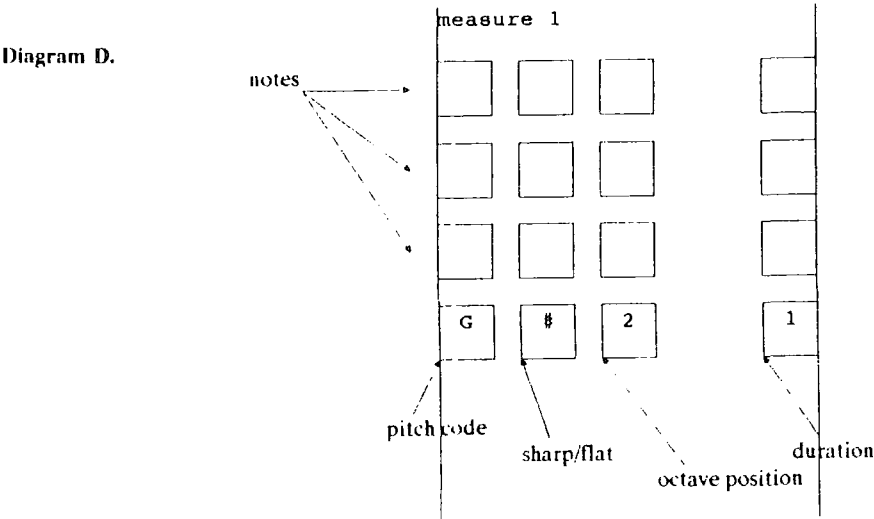
The longest duration, from which all others can be generated as integral multiples, is given the value one. All other durations are specified relative to this basic duration unit. For example, a composition comprising semiquavers, crotchets and minims would have a semiquaver coded as one, a crotchet coded as four, and a minim coded as eight. The rhythm unit used as a basis for coding is not necessarily the shortest duration in the composition. For example, a crotchet may be the smallest actual duration in a composition, but the presence of dotted crotchets would make a quaver a more sensible base unit that the crotchet because the quaver is a common denominator between the crotchet and the dotted crotchet.

Diagram E.

measure 12	
B2	3
A2	6
G♯2	1
F♯2	1
E2	1
measure 13	
F♯2	3
B2	3
E2	3
rest	3
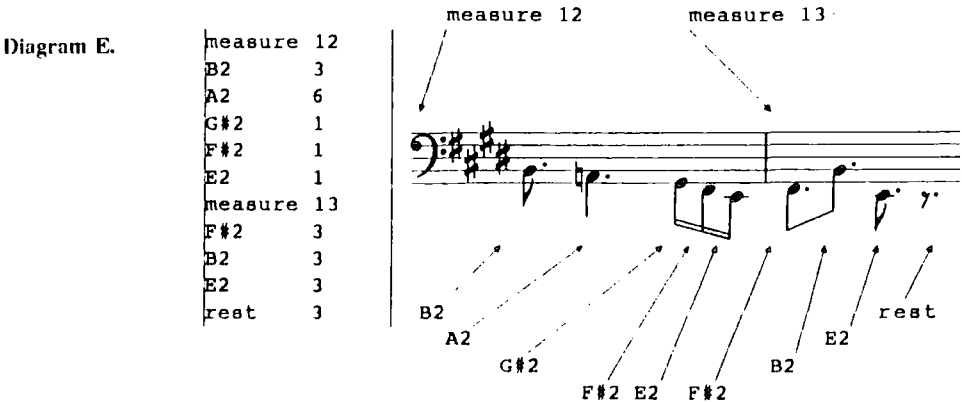
Diagram E is an extract from the fourth fugue of Bach's book two ("Well Tempered Clavier") showing the lowest of three parts. The data represents the notes in bars twelve and thirteen. The pitch C4 always represents middle C. It should be noted, however, that a new octave starts on C and not A as one might expect. This means that the A above middle C is coded as A4 and not as A5. Thus B2 and A2 represent the pitches B and A on the penultimate line and last space of the bass clef respectively. The smallest duration in the fourth fugue is a semiquaver (represented as the value one) and thus the durations three and six represent a dotted-quaver and a dotted-crotchet respectively. The rest in bar thirteen is therefore a dotted-quaver rest. Notice that although the key signature of the fugue is four sharps, F and G sharp have still been encoded onto individual notes of bars twelve and thirteen.

Tied notes are usually represented as single notes. However, if the tie is across a bar, the duration code is suffixed with a minus sign as in the following example (diagram F):

Diagram F.

```
F#2        1
B2         3-
measure 12
B2         3
```


A seven-line header section appears before the first bar of each part. The header contains such information as BWV number, part number, total number of parts in composition, total number of bars, key signature, time signature and clef.

header

```
BEGIN
873 2
Bischoff
1883/1884

3  3
71  4  12  4
12  16  1  0  2
measure 1
C#3        1
B#2        1
```
musical data
```
C#3        1
D#3        1
C#3        1
D#3        1
```


2. 873   2   □   □
    ↑   ↑ | |
    BWV number   movement   number of sec- section number
                       number   tions (blank if (blank if only one)
                                    only one)

3. Bischoff   •
4. 1983/1884   ←------------   Information on source
5.

6. 3   3
    ↑   ↑
    total number of parts   part number

7. 71   4   12   4
    ↑   ↑   ↑   ↑
    number of   key signature maximum number   maximum num-
    bars   (-7=seven flats, of notes per bar ber of notes per
                7=seven sharps)   crotchet

8. 12   1   0   2
    ↑   ↑   ↑   ↑
    time signature clef   mode   movement type
    12/16   (0=treble,   (0=major,   (1=prelude, 2=fugue)
                1=bass)   1=minor)

The Hewlett data is converted into DARMS for use within the
Analysis Environment. Since the description of the Analysis Environment tools
(chapter four) and the test analysis itself (chapter five) generate many
examples of output in DARMS notation, the following pages comprise a brief
outline of the essential elements of DARMS. Although DARMS is complex, yet
thorough, it should be noted that only a subset is used for the
Analysis Environment outlined in this thesis.

DARMS was intended to be used by non-musicians and, consequently, pitches are not encoded using their normal letter names. Pitches are referred to by their position on the stave. Middle C, for example, is represented by the number nineteen.
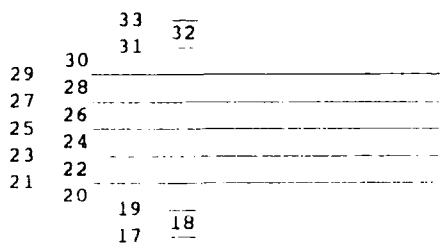
The following musical extract would be encoded as shown:

23 25 26 25 24 22 23
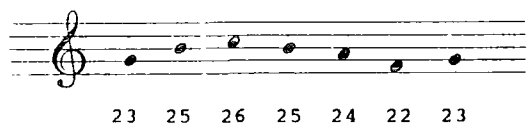
A duration code suffixes the pitch code. Duration codes are represented by the first letter (capital) of the duration name (American, ie a crotchet is a quarter note and thus a Q). Dotted durations should be followed by a dot.

Whole note	W
Half note	H
Quarter note	Q
Eighth note	E
Sixteenth note	S
Thirty-second note	T

The following musical extract, therefore, would be encoded as shown:

23Q 25E 26E 25Q 24Q. 22E 23H

Accidental codes should be placed between the pitch code and the duration code. The following are the permissible accidental codes:

flat	-
double flat	--
sharp	#
double sharp	##

In DARMS, an asterisk is normally used to represent a natural. Notes within the data for the <u>Analysis Environment</u>, however, are always assumed to be natural unless an accidental appears in front of them, so the DARMS asterisk natural-code is not needed. The following musical extract, therefore, would be encoded as shown:

23Q   25-E   26E   25Q   24Q.   22#E   23H

The above extract would be encoded as '23Q 25-E 26E 25*Q 24Q. 22#E 23H' in official DARMS.

Placing the '/' character in the data indicates a bar line. Two bar line symbols '//' represent a double bar and thus the end of the data or a section in the data. The following musical extract, therefore, would be encoded as shown:

23Q   /   25-E   26E   25Q   /   24Q.   22#E   /   23H   //

The letter R represents a rest and should replace the pitch code of a normal note. Thus, a crotchet rest would be encoded as RQ (rest, quarter note).

Although this is only a tiny subset of the DARMS encoding language, it is all that is necessary for the <u>Analysis Environment</u> outlined in this thesis. Since the <u>Analysis Environment</u> tools are created in a modular fashion, new tools can

84

easily be created to examine other details of the score and thus more complex

DARMS data. The following example shows the same extract of music encoded

in "canonical" DARMS and also in the subset used within the

Analysis Environment.

Canonical

```
23!G  !K1-  !M12:8  (29E  30E  29E) 33Q  31E
(29E  28E  29E)  RQ  28E  /  (30E  29E  30E)
26Q  30E  (31E  30E  29E)
```

Subset

```
29E  30E  29E  33Q  31E  29E  28E  29E  RQ
28E  /  30E  29E  30E  26Q  30E  31E  30E  29E
```


Notice the '23!G' code in the canonical DARMS. This is an indication of

what clef is being used. The staff positions used for encoding pitch are the

same no matter what clef is positioned at the beginning of the staff. The pitch G

on the first line of the bass clef and the pitch E on the first line of the treble clef

are both represented by the number twenty-one—only the clef specified at the

beginning of the canonical DARMS data can distinguish the two. The DARMS

data produced by the *darmstrip* and *htod* tools (described in chapter four) uses

only the treble clef and does not, therefore, require a clef code. The pitch G on

the first line of the bass clef has the DARMS treble clef equivalent of 9, ie six

leger lines below the treble clef. Data typed in by hand, for use in the

Analysis Environment, should at present relate everything to the treble clef. The

following musical extract, therefore, would be encoded as shown:


```
12W  13Q  14Q  15Q  13Q  /  14Q  12Q  16W  15H  /  15H  14H  13H.  13Q  /  12W
```

85

Tied notes are encoded in DARMS by suffixing the duration code, of the first of the two tied notes, with the letter J (meaning Join). The following musical extract, therefore, would be encoded as shown:

25-E   26EJ   /   26E   25E
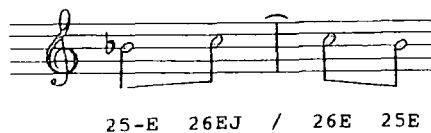
The previous pages have been an introduction to the Hewlett encoding language and a subset of the DARMS encoding language, both of which may be used in the Analysis Environment (to be outlined in detail in the next chapter). A full explanation of the Hewlett method requires only a handful of pages since its rules are few and its complexity minimal. DARMS, on the other hand, is infinitely more elaborate, and although this chapter has formed an introduction to the DARMS subset, the subset is fully expanded upon in context in chapter five.

The Analysis Environment

A computer operating-system is a collection of programs which control the overall operation of a computer system. Operating systems normally contain commands to control the execution of tasks at specific times, the flow of data in to and out of the system, and the amount of processing time allotted to users. An operation can rarely, if ever, be performed on a computer without the assistance of an operating system.

The UNIX operating system started as an experiment in computer science. It was developed by experts for their own use and was intended to allow a small group of users to communicate with each other and to share files and data. It purported to offer a program or tool to accomplish anything a user desired. Text processors, electronic mail, spelling checkers, even games were all available within the standard UNIX package. It was, however, really intended to be a convenient system for supporting computer-program development. The first release looked like the result of a research project (which it was) rather than the result of a development effort and a viable consumer package.

Since the 1970s, more than twenty variants of the original UNIX operating system have appeared as the concept has gained popularity. The operating system is popular with university and industrial environments. Users appreciate its expandability, portability, and standard syntax for the majority of commands. All the commands have evolved over the years to follow a more consistent command syntax. Each command was designed to undertake only one task, but achieve it simply, quietly (ie without messages and prompts), and very efficiently. UNIX is portable and can run on a variety of machines ranging from the small personal computer to the large mainframe computer. Efforts have been made to agree on a standard UNIX which will run on any computer.

The most significant thing about UNIX though, is that problems can be solved and applications created by interconnecting a few simple parts. Users can route the output of one program directly into the input of another, facilitating the solution of large-scale programming problems by combining available small programs, rather than developing completely new programs. The Analysis Environment is designed to make the most use of this UNIX facility. The MSDOS environment provides a similar facility, but only one user can access the computer at any one time, and no file and data sharing is possible without the aid of third-party hardware or software.

One of the most important contributions of the UNIX operating system to the world of computers, is the concept of the 'pipe'. A pipe is an open file connecting two processes, where a process is a command (or 'tool') which is currently running. A user may create a 'pipeline' by connecting several processes together in a linear fashion, through the use of pipes. The pipeline is specified by a series of program names, separated by vertical bars. The program names are the names of tools, and the vertical bars are known as pipe symbols. The output of the program on the left of a pipe symbol is used as the input to the program on the right of the pipe symbol. There are no major restrictions to the length of a pipeline. The concept of a pipeline is fundamental when creating a sophisticated tool out of simpler tools—a more restricted form of pipes and pipelines has consequently been implemented in MSDOS.

Determining the number of users currently using a UNIX system can be accomplished by feeding the output of the system's 'who' command into the command which determines the number of lines in its input.

The command line:

```
$ who | wc -l
```

causes the output of the command 'who', which might appear as:

```
clive          console          Oct 12 09:30
elaine         tty01            Oct 12 09:31
```

to be used as the input for the 'wc' command (the 'word count' command). The '-l' option indicates that only the number of lines is to be printed. Thus, when the whole command line is typed in, the number of users is printed. (The dollar, in the example below, represents the computer's prompt for further input).

```
$ who | wc -l
2
$
```

The user can create a file containing the command line, and can name the file 'users'. By typing 'users', the commands inside the 'users' file will be executed, and the number of users currently using the system, will be displayed.

The tools within the Analysis Environment can be connected together in a similar fashion, to form a pipeline. They may even be fastened to the standard tools already contained within the UNIX operating system.

The command line:

```
$ htod b2f2p1.hew | density -s
```

causes the output of the tool *htod* to be used as the input to the *density* tool. The *htod* tool converts the Hewlett-encoded data file 'b2f2p1.hew' into DARMS data, and might produce the following encoded musical extract:

```
R / R 26 25- 26 27 23 26 25- 24 //
```


R / R 26 25- 26 27 23 26 25- 24 //
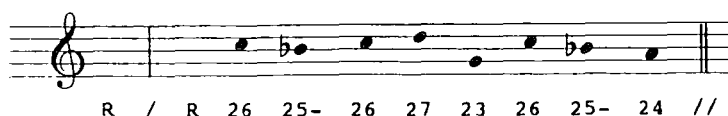
The encoded musical extract is used as the input to the *density* tool. The *density* tool, by default, prints out the total number of pitches in the composition,

and the '-s' option suppresses the output of the DARMS data at the end of the pipeline. Thus, the complete command line prints the total number of notes in the data as follows:

```
$ htod b2f2p1.hew | density -s
Piece density: 8
$
```

Most UNIX and <u>Analysis Environment</u> commands take their input from an 'input stream' and write their output to an 'output stream'. Unstructured sequences of characters used as input or produced as output are referred to as streams, because they have no structure. A command's input stream is called 'standard input' and its output stream is called 'standard output'. Usually, the standard input for a command comes from the terminal's keyboard, and its standard output goes to the terminal's screen. Both standard input and standard output, however, can be redirected from and to files. For example,

```
$ score b2f2p1.dms
```

sends its output to 'standard output' and so the score produced by the *score* tool is displayed on the terminal's screen. The command line:

```
$ score b2f2p1.dms > b2f2p1.scr
```

uses the > character (right angled bracket) to redirect the standard output to the file 'b2f2p1.scr'. The file 'b2f2p1.scr' will be created if it does not exist, or will be emptied before use if it does exist. When the command finishes, the file 'b2f2p1.scr' will contain the score produced.

The >> operator (double angled bracket) also redirects the standard output of a command to a file, but appends the output onto the end of the file instead of overwriting the original contents of the file.

The majority of UNIX and _Analysis Environment_ commands adhere to the following syntax:

```
$ command options filename
```

where the options are usually a series of characters preceded by a minus sign. The options only change the output produced by the command, and do not normally alter the way in which it works. If a filename is missed off the command line, the command reads its input not from a file, but from the keyboard. Using this method, it is possible to try commands on small melodies typed in by the user, rather than complete scores stored in data files.

During the building of a prototype it is often necessary to simplify the design wherever possible, to provide a working model on which tests may be carried out, and results studied. Naturally, increasing the number of simplifications, will create more failings in the prototype model. Similarly, to achieve a working _Analysis Environment_, simplifications are inevitable.

Many arguments have been made for and against the relative significance of pitch and rhythm in music, and although music compositions cease to be music compositions if either pitch or rhythm is removed, the former is initially of more immediate interest. The _Analysis Environment_, therefore, concentrates mainly on the analysis of pitch, accounting for any octave. Another simplification, and thus theoretically a failing, is the disregard of two-dimensional construction—ie the _Analysis Environment_ only contains tools for the analysis of single-line melodies.

All of the _Analysis Environment_ tools operate on data files which contain DARMS data. The DARMS data used for the _Analysis Environment_ data files, however, is a subset of the complete DARMS encoding language, and may only include pitch, rhythm (which is optional), bar lines and rests. The test data used for evaluating the _Analysis Environment_ was chosen, not simply because

of its historical importance, but because an encoded form already existed in the Oxford Text Archive. Many scholars have spent an inordinate proportion of their research time either creating a new music encoding language, or encoding the music scores prior to analysis, and avoidance of this trap in the current context was a priority. Unfortunately, the test data (Johann Sebastian Bach's "Well Tempered Clavier", Book II) had been encoded using a method devised by Walter Hewlett, which differs substantially from DARMS. DARMS though, already established and popular, provides a sensible encoding standard for an environment which is also intended to become a standard.

The first tool in the Analysis Environment then, is the *htod* tool, which converts a Hewlett-encoded data file into a DARMS-encoded data file. By default, the *htod* tool does not include rhythm data since most of the Analysis Environment tools deal specifically with melody. If rhythm is important, specifying a '-r' option on the command line will include rhythm data during the conversion process. Key signatures do not appear in the DARMS data which is produced, and are instead, encoded as accidentals. The transfer of the key signatures to accidentals simplifies processing of the data. DARMS-encoded data files for use in the Analysis Environment should contain a single monophonic line which may or may not include rhythm. Likewise, Hewlett-encoded data files, which are to be converted to DARMS, should only contain a single monophonic line.

Two options may be utilised with many of the Analysis Environment tools. Generally, when a tool has finished its task, the resulting melody (if the

tool has altered the original in some way[99]) or the original melody (if it is unaltered) will be sent to the standard output (ie the screen). Occasionally, when a user only requires information produced during the use of the tool, and not the altered or unaltered melody, it is desirable to suppress the output of this melody. Suppression of the output from a tool can be achieved by specifying the '-s' option on the command line. For example, the command line:

```
$ key -s b2f2p1.dms
```

determines the overall key of the data file 'b2f2p1.dms' and displays the evaluated key on the screen[100], but does not print out the contents of the data file since the standard output has been suppressed and cannot therefore be piped into another tool.

The second, commonly available option, prints out informative messages during the execution of an <u>Analysis Environment tool</u>. The '-v', or 'verbose mode' option, is intended to keep the user informed about what is happening. Used with the *htod* command, it produces the following output to show its progress during the data conversion:

```
$ htod -rv b2f2p1.hew > b2f2p1.dms
Size of data: 2180
Removing BEGIN and END...
Removing header...
```

---

[99]   Two types of tools exist within the proposed <u>Analysis Environment</u>. The first type of tool, known in UNIX as a filter, reads in a DARMS-encoded melody. The tool does a specific job on the melody (such as removing all the repeated notes), and writes out the modified melody ready for use by another tool if required. The second type of tool reads and writes the same melody (ie it does not alter the original), and merely provides information on the melody.

[100] Information generated by a tool (such as the key of the data), as opposed to the original or altered data itself, is actually sent to the standard error stream instead of the standard output stream. This means that although the standard output can be suppressed using the '-s' option to avoid displaying the DARMS data, any information generated by the tool will still be seen because the standard error stream remains unsuppressed and will be displayed on the screen.

```
Replacing measures...
Replacing rests...
Starting conversion...
$
```

To make the <u>Analysis Environment</u> tools more versatile, some of the tools accept a large number of options. To obtain a list of available options, the '-?' option may be used with any <u>Analysis Environment</u> tool. Using the '-?' option with the *htod* tool, for example, displays the following:

```
$ htod -?
Options:
v - verbose mode
r - rhythmic information
$
```

Invariably, one does not normally wish to examine an entire data file at one go. Typically, only a part of the file is of interest. As such, the *extract* tool enables a user to extract a specified part of a DARMS-encoded data file. Melodies may be extracted by choosing a starting note or bar and a terminating note or bar. Omitting the terminating note or bar, extracts a single note or bar. For example, the command line:

```
$ extract -b3 b2f2p1.dms
```

extracts the third bar from the data file 'b2f2p1.dms', and produces:

```
25-Q 24-Q 23E 26E 22Q
```



```
25-Q    24-Q    23E    26E    22Q
```

whereas the command line:

```
$ extract -n5,8 b2f2p1.dms
```

extracts the fifth, sixth, seventh and eighth notes from the data file 'b2f2p1.dms', and produces:

```
23E 26E 25-S 24S
```


23E   26E   25-S   24S
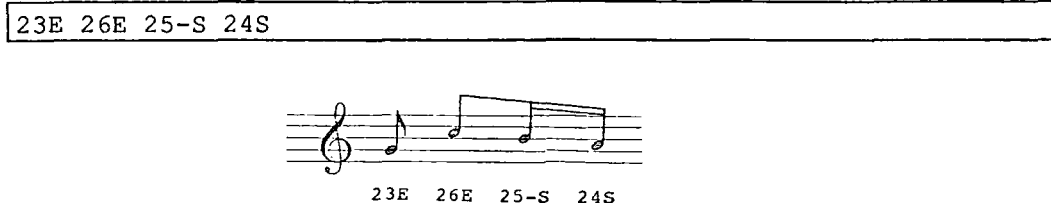
Analogous to the solving of an enigma, music analysis appears to rely heavily upon intuition. An often successful way to solve the majority of complicated puzzles is to make intuitive leaps during the formulation of a solution. However, since intuitive leaps are difficult to quantify mathematically, one might speculate that a computerised version of a specific analytical method is not possible. Although many attempts have been made to computerise previously devised analytical methods, methods specifically designed with computer-aid in mind are better able to capitalise upon the particular advantages to be gained from using the computer as a tool for analysis. Computers are good at repetitive tasks, manipulating numbers and performing calculations. An analytical methodology which has been conceived to involve numbers, and repetitive tasks involving the numbers, must be easier to implement on a computer than one which requires intuitive-type comparisons of musical material. Lengthy searches of similar data—prone to human error from monotony—can be achieved in a fraction of normal time using a computer, and an analytical methodology involving such laborious searches will benefit from the speed of a computer. Computers can 'learn'. Information gleaned from an analysis may be stored in a database[101] and re-used at a later date. The 'old' information is not lost, and can be referred to during future analyses.

---

[101] A database is a highly organised file of related information. Information in the database is usually allocated to a specific subject and is accessible from multiple computer programs.

Correlation and sorting of information is also achieved quickly and efficiently on computer, and many standard algorithms are available.

Since its analytical method is based upon uncomplicated rules and formulae which can be processed, Allen Forte's "Set Theory"[102] may be regarded as a good example of a successful computer implementation. Forte's system, however, was originally conceived for analysis of serial composition, and is regarded by some as unsuitable for tonal music analysis because its results and conclusions refer to melodies which are mathematical in nature and often not intuitive or 'inspired'. To use the subjective labels of suitable, unsuitable, success and failure though, ignores the point behind analysis—ie that of providing a user with fresh insight into a composition's structure. The careful nurturing of a new-born analytical environment, akin to Forte's baby (weaned from raw rules to powerful equations) might offer a route to the successful use of a computer as an aid to tonal music analysis.

What is analysis? Dictionaries define analysis as the resolution of an object into simpler elements, with the intention of finding and showing the object's structure. A tonal composition can indeed be separated into simpler constituent parts or elements. This 'resolution' process, if carried too far, would break the composition up into the smallest musical elements available, ie a collection of pitches belonging to the chromatic scale.

Such an analysis, in its crudest and most basic form, can be regarded as the disintegration of a composition. However, somewhere during this disintegrating or 'decomposing' process there must be a point beyond which the resulting analytical elements are so small as to constitute an unhelpful rotted-representation, and before which the elements are so large as to be

---

102   Forte, A., The structure of Atonal Music, Yale University Press, 1973

simply an abridged version of the original composition. This, for want of a better term, 'breaking-point', is the point in a reductive analysis which will provide the most useful information regarding a composition's actual structure and an insight into its method of creation.

One of the main questions to arise from such a suggestion is that of element size. What is the size of each element at the breaking point? Two pitches, a chord, a bar, maybe an entire phrase? How far can an element be decomposed before it loses its identity, before it can no longer belong to the breaking point? Beethoven's fifth symphony is famous for its 'fate knocking at the door' two-pitch opening rhythmic element. The Beethoven element does not, however, rely solely upon pitch, but instead combines both rhythm and melody to give the element its haunting identity. Remove the pitch, remove the rhythm or change the size (ie length) of the Beethoven element, and the new element lies beyond the analytical breaking point—it loses its identity.

Two tools in the Analysis Environment help to reduce (ie decompose) a composition. The reduction, however, is undertaken in a controlled manner, and should not take the composition beyond the 'breaking point'. Some compositions will undoubtedly lose their 'identity' no matter what is taken away, but this theory will be tested to some extent when the Bach data is analysed using the tools of the Analysis Environment, in chapter five.

The first tool, *rnote*, removes repeated notes from a composition. By default, the *rnote* tool will scan a data file only once, progressing through by a pair of consecutive notes at a time, removing a note if the pair of notes are the same. What this means is that a single pass through the composition is not guaranteed to remove all the repeated notes, and the '-r' option should be employed to force the *rnote* tool to scan, reiteratively, the data file until all repeated notes have been removed. With a data file, called 'demo.dat', containing the following DARMS extract,

```
25  25  /  25  24  25  26  27  /  27  26
```
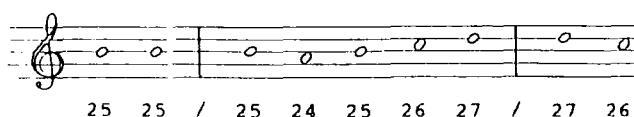


```
25  25  /  25  24  25  26  27  /  27  26
```

the ensuing example shows the result of using the *rnote* tool with and without the '-r' option:

```
$ rnote demo.dat
25 / 25 24 25 26 27 / 26
$
```
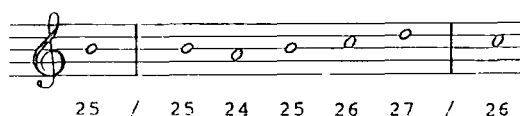


```
25  /  25  24  25  26  27  /  26
```

```
$ rnote -r demo.dat
25 / 24 25 26 27 / 26
$
```



```
25  /  24  25  26  27  /  26
```

The *anote* tool also uses the Schenker approach and removes auxiliary notes from a composition. Again, the *anote* tool will scan a data file once only, and the '-r' option should be used to guarantee the removal of all auxiliary notes. Auxiliary notes are selected pictorially rather than harmonically. This means that if in a set of three notes, the first and last notes are on the same line, the middle note will be regarded as an auxiliary note if it is in the space above or below the other two notes irrespective of what accidentals might be in front of the middle note. The first and last notes, however, must be the same pitch. With a data file, called 'demo.dat', containing the following DARMS extract,

```
25 / 24 25 26 24 25 24
```
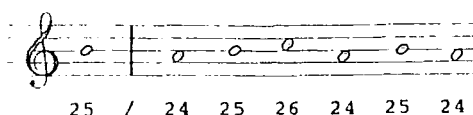
25 / 24 25 26 24 25 24

the ensuing example shows the result of using the *anote* tool with the '-v' (verbose) option:

```
$ anote -v demo.dat
Size of data: 24
Removing...
25-24-25
24 removed
26-24-25
24-25-24
25 removed
25 / 25 26 24 24
$
```


25 / 25 26 24 24
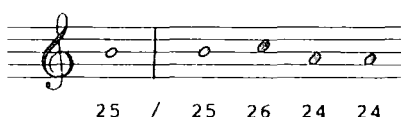
The size of the data displayed in the previous example is the number of characters (including control characters such as a carriage-return) in the data file itself. The rest of the output produced by the '-v' option shows each set of three notes being examined in turn, and indicates when an auxiliary note has been removed.

The *rnote* and *anote* tools may be used together since the removal of repeated notes might produce new auxiliary notes, and the removal of auxiliary notes might produce new repeated notes. Both tools may be bolted together in the standard UNIX fashion:

```
$ extract -b1,3 b2f2p1.dms | anote -r | rnote -r
```

The above command line would produce the following three bars, from the data file 'b2f2p1.dms' (which contains the first part from Bach's second fugue of book two (WTC)), after the *extract* command,

```
R / R 26 25- 26 27 23 26 25- 24 / 25- 24- 23 26 22 /
```



```
R / R 26 25- 26 27 23 26 25- 24 / 25- 24- 23 26 22 /
```

the following output after the *anote* command,

```
R / R 26 26 27 23 26 25- / 25- 24- 23 26 22 /
```



```
R / R 26 26 27 23 26 25- / 25- 24- 23 26 22 /
```

and the following final output after the *rnote* command:

```
R / R 26 27 23 26 25- / 24- 23 26 22 /
```



```
R / R 26 27 23 26 25- / 24- 23 26 22 /
```

It should be noted that only the final output (ie the output at the end of the pipeline) is sent to the standard output stream (ie the screen).

Any data, produced from an Analysis Environment command line, will be in DARMS format. If a scholar is not conversant with DARMS, output in that format will not be of tremendous use. Fortunately, the Analysis Environment provides two tools for converting DARMS output into a more 'musical' format. Although the first tool, *score*, requires nothing special in the way of output devices, the second tool, *play*, will only run on an MSDOS machine[103] (ie IBM

---

[103]   MSDOS contains a restricted version of the UNIX pipe facility. Since the tools of the analysis environment are all written in the C programming language, they can be compiled for an MSDOS machine. The Analysis Environment, therefore, will run on an IBM compatible computer, and moves its application from simply an academic environment to a personal or home environment. Using the Analysis Environment under

Personal Computer compatible) since it requires the standard built-in speaker to reproduce its audible monophonic-output. The *score* tool uses the standard character set to produce a pseudo-music notation. Rhythm is not displayed, but any scholar with a slight knowledge of music notation should be able to decipher the output of the *score* tool. Specifying the '-v' option (normally 'verbose mode') will print bar numbers in the appropriate places.

The *play* tool (only for use under the Microsoft Disk Operating System—MSDOS) produces an audible rendition of DARMS data fed in on standard input. This means that any command line which produces DARMS output can have the *play* tool bolted onto the end, and the result will be played for the scholar to hear—rather like the intentions of Keller. An '-r' option allows the *play* tool to read and utilise the rhythm data. The overall tempo of the piece is calculated automatically by searching for the shortest note within the data and using that as a base duration—similar to preparation for music sight-reading, which entails finding the 'blackest' (ie densest) part of a score, and calculating an overall tempo from that. Initially as a 'trivial' option, the '-n' option plays every note in a stream of DARMS data with a nominal zero seconds duration. This produces a 'noise', and is probably the audible shape of piece squeezed into a matter of seconds. It could have a future use in the comparison of entire melodic lines. If the *extract* tool is used to extract the first three bars of a data file, and produce the following DARMS,

---

MSDOS, however, is restrictive because none of the hundreds of standard UNIX tools or shellscript programming languages (of which the Bourne shell programming language will be expanded upon in chapter five, and is outlined in appendix C) are available. The research outlined in this thesis has taken place in a UNIX environment, but has also been tested in the more restrictive MSDOS environment. Only one tool, the play tool, is exclusive to the MSDOS environment.

```
26 25- 26 27 23 26 25- 24 / 25- 24- 23 26 22 /
22 21- 20 23 22 21- 20
```
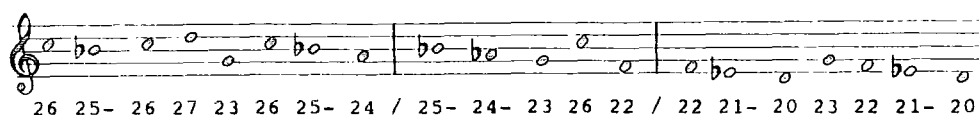

the *score* tool, when bolted on to the end of the *extract* tool, will produce the following output:

```
$ extract -b1,3 b2f2p1.dms | score
+----------+-------+--------+
|          |       |        |
+----O-----+-------+--------+
|O  O  O   |    O  |        |
+-bO----bO-+bO-----+--------+
|        O| bO     |        |
+-----O----+----O--+----O---+
|         |      O|O    O   |
+---------+-------+-bO---bO-+
                  O    O
$
```
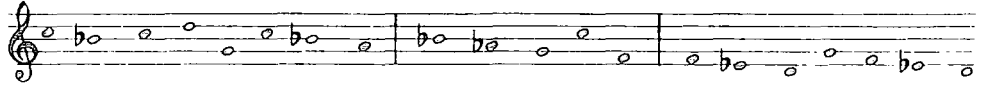
Even when DARMS output is piped into the *play* tool, the output still comes out of the other side and is available to go into another tool. This means that sound output can be produced at different points along the pipeline, (before and after removing repeated notes for example). The following example plays the contents of a data file called 'b2f2p1.dms' before and after the *rnote* and *anote* tools have been used to remove the repeated and auxiliary notes respectively.

```
$ play b2f2p1.dms | rnote -r | anote -r | play
```

Tools that modify musical data might equally be beneficial to composers or those simply interested in manipulating sound. Although the majority of tools in the <u>Analysis Environment</u> perform tasks related to music analysis, three of the tools transform data, generating new structures and, when piped through the *play* tool, new sounds. The first of these 'composing' tools, *shuffle*, changes the order of bars within a melody. By default, the *shuffle* tool puts complete bars of

a given melody into an arbitrary order. For example, given the following melody:

```
26 25- 26 27 23 26 25- 24 / 25- 24- 23 26 22 / 22 21- 20 23 22
21- 20
```


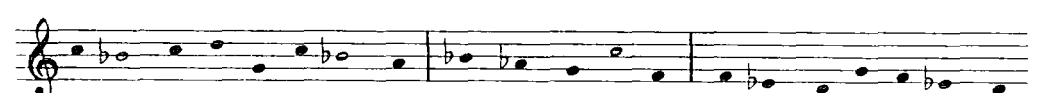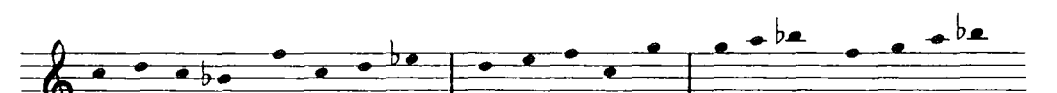26 25- 26 27 23 26 25- 24 / 25- 24- 23 26 22 / 22 21- 20 23 22 21- 20

the *shuffle* tool might rearrange the bars to produce the following:

```
25- 24- 23 26 22 / 22 21- 20 23 22 21- 20 / 26 25- 26 27 23 26
25- 24
```


25- 24- 23 26 22 / 22 21- 20 23 22 21- 20 / 26 25- 26 27 23 26 25- 24

The '-n' option of the *shuffle* tool specifies the size of the note unit to be shuffled, overriding the default size of a bar. For example, specifying '-n3' will shuffle units of three pitches in length into and arbitrary order. If the above three bars are subjected to the *shuffle* tool with the option '-n4', the following data might be produced:

```
25- 24- 23 26 23 26 25- 24 23 22 21- 20 22 22 21- 20 26 25- 26
27
```


25- 24- 23 26 23 26 25- 24 23 22 21- 20 22 22 21- 20 26 25- 26 27

The '-c' option, when used with the *shuffle* tool, ensures that the end of one shuffled unit moves smoothly onto the beginning of the next shuffled unit. A smooth move for one unit to another is deemed to a repetition of a pitch or a progression by up to a major third. Bars that cannot be shuffled smoothly are

discarded. Shuffling the same three bars with the options '-c' and '-n2' might produce the following:

```
21- 20 21- 20 22 22 23 26 25- 24- 23 22 23 26 25- 24 26 25- 26
27
```



```
21- 20 21- 20 22 22 23 26 25- 24- 23 22 23 26 25- 24 26 25- 26 27
```

The *squash* tool can be made to augment or diminish the size of intervals in a given melody. By default, the tool diminishes all intervals by a semitone. Using the *squash* tool, with no options, on the following melody:

```
26 25- 26 27 23 26 25- 24 / 25- 24- 23 26 22 / 22 21- 20 23 22
21- 20
```



```
26 25- 26 27 23 26 25- 24 / 25- 24- 23 26 22 / 22 21- 20 23 22 21- 20
```

produces:

```
26 25 26 27- 23 25 25- 25- / 25- 24 24 27- 23 / 23 23- 23- 25-
24 24- 24-
```



```
26 25 26 27- 23 25 25- 25- / 25- 24 24 27- 23 / 23 23- 23- 25- 24 24- 24-
```

Since the interval of unison cannot be reduced, note repetition always remains as note repetition. The '-a' option, when used with the *squash* tool, augments intervals by a semitone instead of diminishing them. The amount of increase (in semitones) applied to the augmentation or diminution may be specified with the '-n' option. For example, if the '-a' and '-n2' options are used with the *squash* tool on the following data:

```
26 25- 26 27 23 26 25- 24 / 25- 24- 23 26 22 / 22 21- 20 23 22
21- 20
```



```
26 25- 26 27 23 26 25- 24 / 25- 24- 23 26 22 / 22 21- 20 23 22 21- 20
```

the following result is produced:

```
26 24- 26 28 23 27 25- 23 / 25- 23- 21- 25- 20- / 20- 17 16- 20-
17 15 13
```



```
26 24- 26 28 23 27 25- 23 / 25- 23- 21- 25- 20- / 20- 17 16- 20- 17 15 13
```

The *vary* tool also affects intervals. The tool reverses the direction of all intervals. Ascending intervals are made to descend, and descending intervals are made to ascend. The '-a' and '-d' options reverse the direction of only ascending and descending intervals respectively. For example, applying the *vary* tool to the following data:

```
26 25- 26 27 23 26 25- 24 / 25- 24- 23 26 22 / 22 21- 20 23 22
21- 20
```



```
26 25- 26 27 23 26 25- 24 / 25- 24- 23 26 22 / 22 21- 20 23 22 21- 20
```

produces the following result:

```
26 27 26 25- 29 26 27 28- / 27 28 29 26 30 / 30 31 32- 29 30 31
32-
```



```
26 27 26 25- 29 26 27 28- / 27 28 29 26 30 / 30 31 32- 29 30 31 32-
```

Music scholars have standard ways of expressing music structure. For example, binary and ternary form can be expressed symbolically, in terms of letters, as AB and ABA respectively. Forte devised a parsing technique which can be used to rewrite music repeatedly in a simpler form until there remains no repetition of musical material. His technique, which was revised and used at a later date by Ian Bent and John Morehen, forms the basis for the *form* tool. By default, the *form* tool takes account of accidentals and octave position of notes. This, more often than not, results in a symbolic form of little help since there are so many different units and often no repetition of any units. Consequently, the '-a' and '-o' options force the *form* tool to ignore accidentals, and the octave position of notes, respectively. This usually decreases the number of symbolic units produced and increases the amount of repetition of the units. By themselves, the symbolic units are of limited used, but a '-k' option produces a key which shows the notes referred to by each symbolic unit. As an example, the first three bars of the first part from Bach's second fugue in book two (WTC) may be extracted from the data file 'b2f2p1.dms' and redirected into the file 'testdata', using the following syntax:

```
$ extract -b1,3 b2f2p1.dms > testdata
$
```

The new data file 'testdata' contains the following DARMS:

```
26 25- 26 27 23 26 25- 24 / 25- 24- 23 26 22 /
22 21- 20 23 22 21- 20 /
```


26 25- 26 27 23 26 25- 24 / 25- 24- 23 26 22 / 22 21- 20 23 22 21- 20
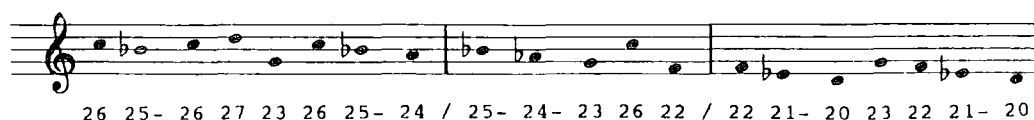
Using the *form* tool on the data file 'testdata' with the '-s' option, to suppress the output of the original DARMS data, produces the following output,

```
$ form -s testdata
A B C D E C F G H G
$
```

which shows that symbolic units '23 26' and '21- 20' are repeated (diagram A).

Diagram A.
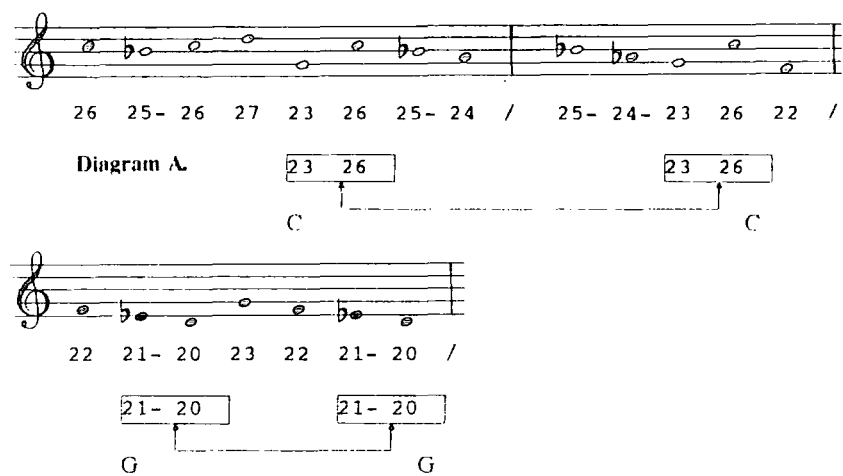
Using the *form* tool with the '-a' and '-o' options, to ignore accidentals and octave position respectively, produces the following output,

```
$ form -aos testdata
A B C D D C E F G F
$
```

which shows that symbolic units '23 26' and '21- 20' are repeated as before, but also that '25- 24' is repeated as '25- 24-' (diagram B).

Some people might argue that resemblance after removal of accidentals is no more significant than any other transposition. Removal of accidentals leaves the 'shape' of the melody and not the exact sound. This enables similar shaped units such as '25- 24' and '25- 24-' to be identified. Some compositions contain fragments that are conceived based on shape and structure as much as sound.

Diagram B.

```
 26  25- 26  27  23  26  25- 24  /   25- 24- 23  26  22  /

Diagram B.      23  26  25- 24      25- 24- 23  26
                        C                          C
                     D               D
```

```
 22  21- 20  23  22  21- 20  /

 21- 20         21- 20
   F               F
```
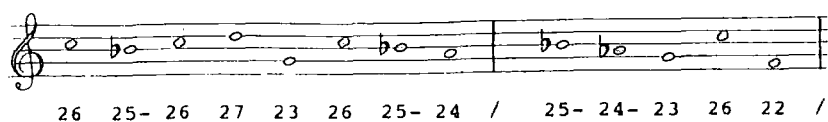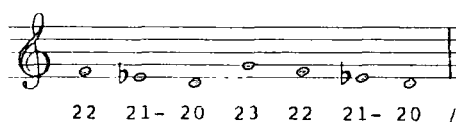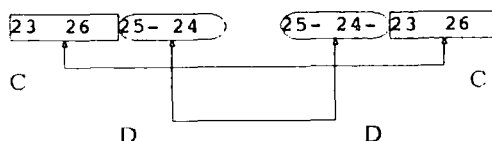
The following example uses the '-k' option to display a key for all the symbolic units:

```
$ form -aoks testdata
Final element size: 2
A = 26   25-
B = 26   27
C = 23   26
D = 25-  24
D = 25-  24-
C = 23   26
E = 22   22
F = 21-  20
G = 23   22
F = 21-  20
$
```

A search for the main building-blocks of a composition, along with the techniques employed in fastening them together, will give more insight into its real structure. Analogous to the building of a house, where utilisation of bricks (as opposed to reliance upon a foundation), defines its structure and appearance, a search for musical structure requires analysis of the melodic and contrapuntal lines created from brick-like notes, (avoiding any temptation to

108

examine the foundation-like harmony). Admittedly, a house with poor foundations might either sink or fall to the ground, but it is the house's initial outward appearance, similar to the hearing of a composition or perusal of a score, which arouses one's imagination, creating vivid imagery.

In psychoanalysis, the interaction of conscious and subconscious elements within the mind are investigated in a bid to bring the latter element into consciousness. Likewise, in 'musicanalysis' [sic], one deals with relationships between the 'initial subconscious idea' and the conscious score. Since the subconscious part of the mind is able to influence all actions, it follows that the subconscious musical idea, similar to the 'brick'—never actually thought about, though frequently used—can influence the compositional process, and must be traceable through fine and methodical study of a work's score. Therefore, the subconscious idea, or 'brick', together with construction plans, may help to reveal a composer's compositional technique, and as such, offer the goal and one of the many different definitions of analysis—the search for a subconscious idea, behind, and within a composition.

The very essence of the house is a single red brick, repeated. However, essentially progressing in straight lines, it, and others, work contrapuntally with higher and lower parts, rarely allowing brick-edges to meet, except at window and door climaxes. The wall-end cadences, emphasised by cleanly cut half-bricks, act as movements by dividing the house into sections; and, with clever use of vertical bricks to emphasise such climactic windows, creates a style unique to the building.

Many of the Analysis Environment tools aid in the search for either an underlying idea, or usage of specified 'building blocks'. The *ursatz* tool is perhaps the most straightforward in its usage. By default, the *ursatz* tool

displays possible locations for 8-1, 5-1, 3-1, standard and prolonged Ursaetze[104]. Since each tool in the <u>Analysis Environment</u> is designed to undertake one specific task and perform it efficiently, the *ursatz* tool does not calculate the key of a DARMS data stream. The key must be specified on the command line, after the '-k' option. G sharp minor, for example, would be specified as G#m, and inserted into the command line as shown below:

```
ursatz -k G#m b2f2p1.dms
```

If only a specific type of Ursatz is required, the options '-8', '-5' and '-3' will select only 8-1, 5-1, or 3-1 Ursaetze respectively. Using the '-p' option selects only prolonged Ursaetze. To select all possible locations for 5-1 and 3-1 prolonged Ursaetze, the '-5', '-3' and '-p' options should be inserted into the command line as shown below:

```
$ ursatz -53p -k G#m b2f2p1.dms
```

The following example will display all possible locations for a 5-1 Ursatz in Bach's second fugue of book two, part one[105]:

---

104   The standard 3-1 Ursatz comprises a melodic and scalic progression from mediant down to tonic with bass progression from tonic to dominant and back to tonic (ie 3-2-1 over I-V-I). 8-1 and 5-1 Ursaetze are similar, but as their titles imply, the melodic progression is from tonic to tonic and dominant to tonic respectively. The bass progression in both instances remains tonic-dominant-tonic. Prolonged Ursaetze are standard Ursaetze in which the melodic progression is interrupted. For example, 3-2-1 might be prolonged to 3-2, followed by 3-2-1.

105   For the purpose of this research, data under analysis was stored in files named using a fixed convention. Data files containing Hewlett data were given the suffix .hew, whilst data files containing DARMS data were given the suffix .dms. The initial part of the filename indicated the book number, fugue number and part number using the format 'b*n*f*n*p*n*', where *n* was replaced by the actual number of the book, fugue and part. For example, the first part of Bach's second fugue from book two (WTC), encoded in DARMS, would be stored in a data file named 'b2f2p1.dms'.

```
$ ursatz -vs5 -k Cm b2f2p1.dms
Key name: Cm, Key number: 8
Verbose mode switched on.
Suppressing output of filtered data.
Searching for 5-1 Ursaetze...
Size of data: 827
Converting to chromatic
Sequence 3, 1, 11, 10, 8.
Found 5-1 at positions: 4, 12, 14, 15, 20
Found 5-1 at positions: 4, 12, 14, 15, 179...
```

Although the output from the above example has been truncated to save space, the positions of the first two suggested Ursaetze are shown in diagram C. Most of the output in the above example is the result of specifying the '-v' (verbose) option on the command line which produces informative messages whilst the tool is running.

Diagram C.
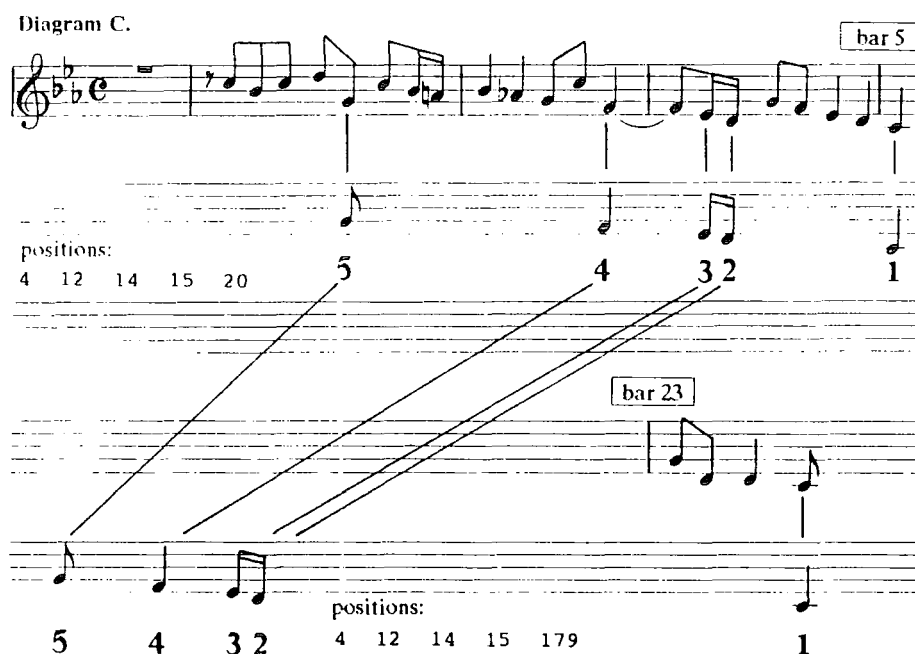
Superficially the *key* tool appears to remove any need for the scholar to determine the key or progressions throughout a piece. It is not, however, guaranteed to find the correct key since its method is entirely mathematical and not in the slightest intuitive. Highly chromatic pieces will cause the *key* tool some problems because even a short melody will contain a large number of extra sharps and flats, and determining what is and what is not superfluous to the current key is difficult to quantify successfully. If no options are specified on the command line, the *key* tool prints out a suggested key for a DARMS data stream. If the *key* tool cannot calculate a 'definite' key from the sharps and flats used, it will describe the key as unknown or display a question mark, and the scholar will have to resort to a manual method. It is important that scholars realise that the tools of the <u>Analysis Environment</u> do not intend to enforce 'the' correct answer, but merely suggest 'an' answer or a set (range) of answers. The '-b' option used with the *key* tool evaluates a key for each bar in a composition—useful if the piece is very chromatic. Progressive mode, invoked with the '-p' option, attempts to show the key progression throughout the piece. The current key is recalculated after each note. If the key has changed, it is displayed. Below, is an example progressive output for Bach's second fugue in book two. The first thirteen bars are extracted and piped through the key *tool*. The major problem with the *key* tool is that it only examines a melody, and not the underlying harmony. The pitch C, therefore, in the top part of the last bar, could belong to a number of keys, and so the *key* tool, because no definite key can be calculated, describes the key for the last bar as unknown.

```
$ extract -b1,13 b2f2p1.dms | key -ps
CF Eb     ? Eb     Ab Eb     ?Fm ?
1   2   3 4 5 6   7 8 9   10 11 12   13
$
```

Diagram D shows how the key progressions, evaluated by the *key* tool in the example above, relate to the actual score.

key signature hard-coded into data

Initial key determined by first note - C major

Bb indicates F major

Ab indicates Eb major

E natural - key unknown

Eb indicates Eb major

Db indicates Ab major

D natural indicates Eb major

E natural - key unknown

Db indicates F minor

A natural - key unknown
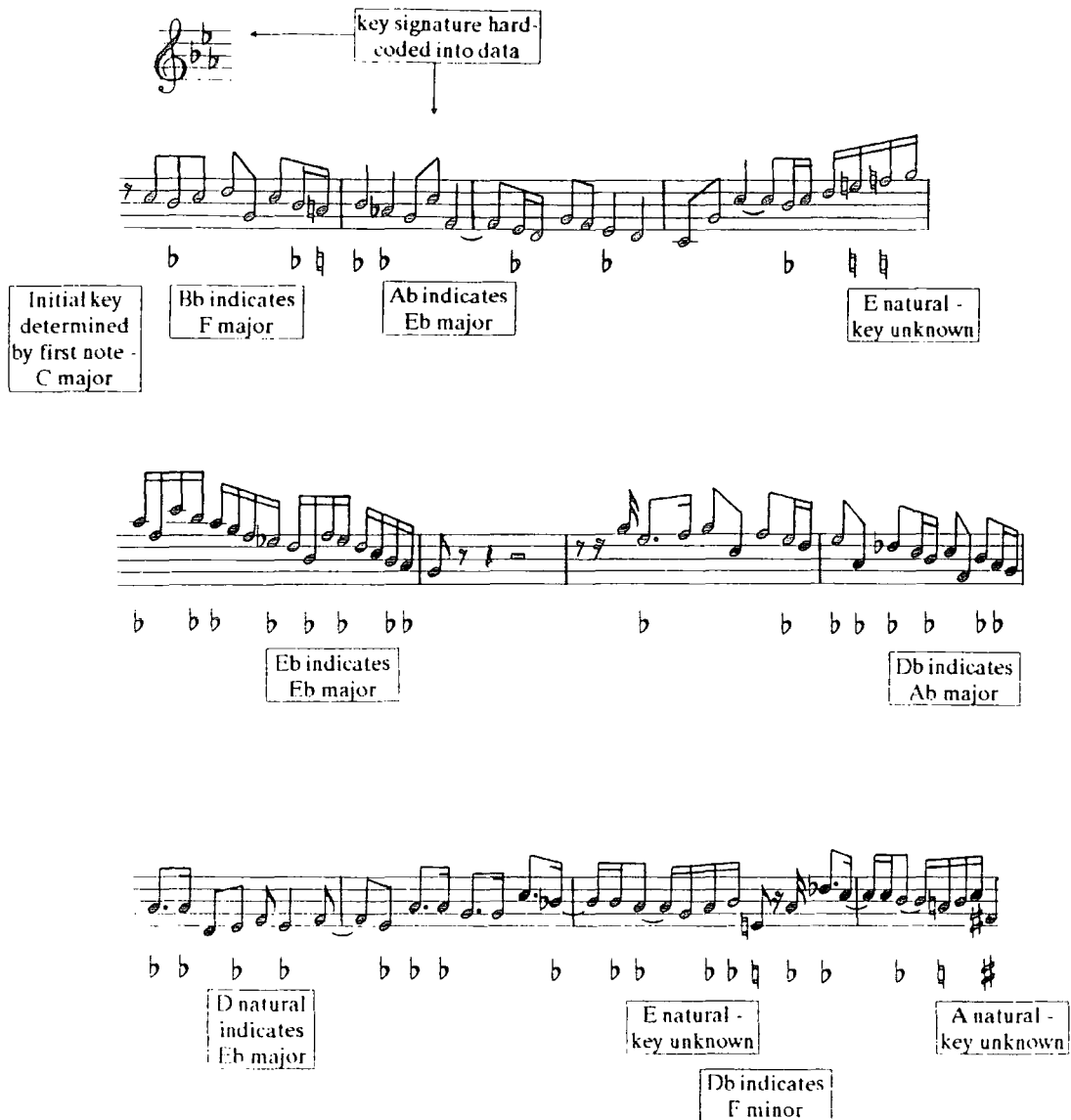
Despite its inadequacies, the information from the *key* tool is still useful. The *key* tool has no problems with standard scales and unadventurous pieces. The very complexity of Bach's compositions though, is what causes the difficulties for the *key* tool. The standard 'echo'[106] tool of the UNIX environment is used in the following example to send a DARMS string of data (an ascending

---

106   The echo tool displays a string of text on standard output (usually the screen). Any string of text, therefore, can be echoed using the echo tool, and piped through another tool.

scale of C harmonic minor) through the *key* tool, and as one would expect, the evaluated key is shown to be C minor:

```
$ echo "19 20 21- 22 23 24- 25 26" | key -s
Key of piece: C minor
$
```

The *cursatz* tool will search a DARMS data stream for a specified sequence of pitches. Like the *ursatz* tool, the *cursatz* tool will locate a sequence of pitches wherever it occurs within the DARMS stream. For example, the pitch sequence "C D E" would be found in the data stream "C̲ G D̲ D G E̲, even though there are pitches in between the original C, D and E. By default, however, the pitches found in the DARMS data stream must be in the same octave position as the pitches in the original sequence. Specifying the '-o' option will ignore the octave position of the original pitch sequence, but will increase the number of located pitch sequences dramatically. The initial fugue subject from Bach's second fugue in book two, may be encoded as the following DARMS string:

```
26 25- 26 27 23 26 25- 24 25-
```

Using the above DARMS string as the 'customised Ursatz', the following output will be produced using the *cursatz* tool:

```
$ cursatz -u "26 25- 26 27 23 26 25- 24 25-" b2f2p1.dms
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 24
1 2 3 4 5 6 7 8 39
1 2 3 4 5 6 7 8 44
1 2 3 4 5 6 7 8 59...
```

The lines of numbers produced are possible locations for the 'customised Ursatz', and are shown in relation to the score in diagram E. Each number in a sequence is a suggested position within the score for the associated note of the 'customised Ursatz'. For example, in the output shown above, the 'customised Ursatz' has been located in the first nine notes of the data, and also in the first

eight notes with the twenty-fourth note forming the last note of the 'customised Ursatz'.

Two more tools can be used as an aid in dividing a composition up into building blocks. The first, *semio*, divides a DARMS-encoded data file into melodies of a specified size. In an effort to account for all building blocks and omit nothing which might be of importance, the melodies extracted from the data overlap. The *semio* tool displays the entire data file, fifteen notes at a time, together with the starting position of each melody and the positions of its other occurrences listed underneath each note of the data file. This enables a melody and its repetitions to be identified quickly and easily, and also highlights melodies which might be more important than others. Extracted melodies default to a length of one note, but this length may be increased using the '-n' option. The following example output identifies the usage of three-note melodies in Bach's second fugue of book two.

```
$ semio -sn3 b2f2p1.ds
_____
8---6---8---10--3---8---6---5---6---
^1  ^2  ^3  ^4  ^5  ^6  ^7  ^8  ^9
_____
1   2   3   4   5   6   7   8   9
23  24  142 184 22  91  92      33
58      166     77              44
        183     185             195.
```

The above output is only a small part of the total output produced by the *semio*
tool. In fact, it shows the results for the first nine notes of the DARMS-encoded
data file. The first line of the output is the data itself, but in a chromatic notation
that does not account for octaves. Zero represents the note E, whilst eleven
represents the note D sharp. Below that line, for ease of reference, are the
numbers (ie the positions) of the notes themselves within the DARMS-encoded
data file. Finally, underneath the second horizontal line, are the starting
positions of three-note melodies identical to the melody starting above each list.
For example, the first column contains the starting positions of the melody
'8 6 8' (C, B flat, C) which are 1, 23 and 58 (ie the first, twenty-third and fifty-
eighth note positions), whilst the second column contains the starting positions
of the melody '6 8 10'. Diagram F shows the output, from the above example, in
relation to the score.

Diagram F.

116

The *motif* tool produces a similar form of output, but only searches for a melody specified by the user, and not for all melodies of a specific length. The *motif* tool lists the entire DARMS-encoded data file, fifteen notes at a time, together with the user-specified melody. The user-specified melody appears under the DARMS-encoded data, showing how it relates to the data. By default, the *motif* tool uses the exact melody specified by the user, but the '-n' option may be employed, allowing the expansion of the melody up to a specified number of notes. This means that a user-specified melody of 'C, Bb, C', and a '-n' limit of four, would match '<u>C</u>, D, <u>Bb</u>, <u>C</u>' and '<u>C</u>, <u>Bb</u>, Bb, <u>C</u>' for example, as well as the obvious 'C, Bb, C'. The following example output from the *motif* tool shows how the first two notes of the opening fugue subject from Bach's second fugue of book two can be located elsewhere in an expanded form.

```
$ motif -n5 -m "26 25-" b2f2p1.dms
   26  25- 26  27  23  26  25- 24  25- 24-
1. 26..25-
2.             26..............25-
3.                     26..25-
4.                     26.........25-
```

The first line, displayed in the above example, shows the data file itself. Occurrences of the motif, expressed on the command line after the '-m' option, are shown below the contents of the data file, and numbered at the left-hand side. The output from the above example is shown in relation to the score in diagram G.

Diagram G.

Having decided that analysis is a search for building blocks, upon which all might be based, the final result of an analysis could be a readable map showing routes from the building blocks, via piece segments, to a final score—in essence a stemmatic chart. This stemmatic chart, with the building blocks (in future to be labelled the 'base melody', although the blocks might well not constitute a melody) at the top and score at the bottom, will account for, and display all notes within the composition.

Analytical methods involve the application of reductive processes on the score. Reductive processes involve a high degree of melodic, harmonic, rhythmic, and pitch comparison. Since there theoretically can only be one base melody, and perhaps one set of true routes from base to score, to achieve such a unique map requires standardisation of the analytical reductive technique. Comparison, and identification of what is, and what is not similar—a wholly intuitive task—is the major reason for large discrepancies between results of like analyses. Standardisation of similarity, creating a representation of this form of intuition which can be processed, should narrow down the variation in results of previous and new analyses, although at the end of the day, the scholar will be left to draw his or her own conclusions from the results.

A computer attempting to imitate human-conceived analytical method is destined for failure, since such analytical method relies heavily upon intuition and this has yet to be modelled in computational terms. A single analytical decision might require several hundred computer operations to model it, and as such, creates an ineffable difficulty when endeavouring to formulate analytical methods such as Schenker, Semiotic, and Thematic—methods which rely upon such intuitive approaches as selection of an Ursatz, reduction, and identification of transformation or similarity. In the Analysis Environment, the tools are not computerised versions of specific analytical methods, but merely parts of the

analytical methods—the quantifiable parts—which may be bolted together in a variety of ways.

A human-analytical process is composed of numerous intuitive steps, and transferring that process to a computer will not only necessitate definitions of all such steps, but will require a computer with phenomenal memory to accommodate so vast a number of representative operations, and tremendous processing power to execute a vast number of operations in a short space of time. Sadly, unlike a single computer program running on various machines, the outcome from an intuitive step is entirely dependent upon the human user, and thus, different humans possessing diverse intuitive powers conjure up unique analyses of the same work. Since the tools can be bolted together in a variety of ways, the output from an analysis will be entirely dependent upon the user. There will, however, be a certain consistency in the results imposed by the limitations of the tools. By definition, if an analysis is undertaken in the 'pursuit of a work's possible structure', the analysis must arguably produce only one solution because the analyst is searching for a single structure. A 'solution', however, could be represented in a number of different ways and in turn have different interpretations.

Reliance upon, or application of intuition will not provide a unique result. To avoid a multitude of controversial results, intuition needs to be either ignored, or evaded by algorithmic simulation. Statistical analysis, since its feature-counting requires no intuitive processes, may be readily and successfully implemented on computer. The intuition, however, is left to the system user, who must fabricate his own relationships and links from the results. This disregard for intuition produces a purely mechanistic analytical system, which simply provides numerical lists and has no ability to imply relationships within data. Thus, for a 'successful' yet 'black box' analysis, intuition should be imitated in some way.

Two tools in the Analysis Environment are designed specifically for counting features. The default output from the tools is a series of lists and numbers. No conclusions are drawn, no intuition is involved. The computer locates and counts methodically, but does not attempt to imply anything from the resulting data. The *freq* tool, without any options specified on the command line, counts the frequency of each accidental used, the total number of bars, the total number of notes, the frequency of each pitch used (accounting and not accounting for octave position), the frequency of each duration used, the frequency of each two-note and three-note phrase. Specifying options on the command line produces output only for those items implied by the options. For example, the following command line,

```
$ extract -n1,10 b2f2p1.dms | freq -aps
```

extracts the first ten notes from the data file 'b2f2p1.dms', pipes the result through the *freq* tool, and then counts the frequency of each pitch used (the '-p' option) and each accidental used (the '-a' option), producing the following output:

```
$ Pitch usage:
Ab 1
A  1
Bb 3
C  3
D  1
G  1
Accidental usage:
Ab 1
Bb 3
```

The ten notes extracted by the *extract* tool are shown below:

It should be noted, however, that the Hewlett encoding method does not employ a special code for key signatures and extra codes for notes employing accidentals which contravene the key signature. Instead, all sharps and flats are coded into the score itself, irrespective of whether or not they are part of the key signature or just genuine accidentals. The '-a' option of the *freq* tool, therefore, actually counts the frequency of each 'black' note and not the frequency of 'true' accidentals.

Standard DARMS uses a special code for the key signature, whilst the DARMS subset used within the <u>Analysis Environment</u> does not. Avoiding the key signature, and hard-coding the sharps and flats into the score does not contravene the DARMS standard, but does restrict the use of <u>Analysis Environment</u> tools on standard DARMS. To circumvent this problem, the *darmstrip* tool strips standard DARMS-encoded data files of all information currently not used by the <u>Analysis Environment</u> tools. Employing the *darmstrip* tool does not 'damage' a data file since its output is sent to the 'standard output' (ie the screen) and must therefore be redirected into another file in order to save the output. Output can be redirected using the following format:

```
$ darmstrip original.dms > b2f2p1.dms
```

The above example strips out all unnecessary data from the file called 'original.dms' and sends the output generated to a new file called 'b2f2p1.dms', creating the new file if it does not exist, overwriting it if it does exist—it does not physically change the contents of 'original.dms'. By default, when the *darmstrip* tool is converting standard DARMS to the subset used in the <u>Analysis Environment</u>, it hard-codes the key signature into the data itself (analogous to the Hewlett data), and removes the key signature from the beginning of the data. Accidentals (which are theoretically any sharp, flat or natural signs on the musical stave) in the data files containing the subset DARMS might not necessarily be 'true' accidentals if they belong to the key

signature. If there is a possibility that these accidentals would never have appeared on the musical stave prior to their hard-coding into the data, they cannot be classified as 'true' accidentals. To circumvent this problem, the key signature will not be hard-coded, during the conversion from standards DARMS to subset DARMS, if the *darmstrip* tool is used with the '-k' option. Data which has been converted using this option will a give true count of accidentals when piped through the *freq* tool. Other tools, however, might procure unexpected results (ie the *play* tool renders a somewhat 'modal' version of melodies with a high number of sharps or flats in the key signature).

The *endan* tool also counts features, but is designed specifically for examination of the end of a composition. The *endan* tool produces a list of statistics based on the last five bars of a composition. The tool counts the notes employed in each of the last five bars, all of the notes employed throughout the last five bars, the repeated notes employed in each of the last five bars and all of the repeated notes employed throughout the last five bars. As well as these features, the '-d' and '-m' options may be used to display the distance between the highest and the lowest notes in the last five bars, and display a small graph showing the melodic movement during the last five bars respectively. The graph produced for melodic movement uses the standard characters '/' and '\' to represent a large intervallic rise and fall, whereas the '.' symbol represents scalic or chromatic rise and fall. Specifying options on the command line produces output only for those items implied by the options. For example, the command line:

```
$ endan -nms b2f2p1.dms
```

displays the total number of notes employed in each bar (the '-n' option) and outputs a graph of melodic movement (the '-m' option), producing the following example:

```
Notes per bar
9/10/5/5/1//
Movement:
..\/....\..\/.......\/.......\...
```

If the analysis of an ending is of major importance to the scholar, the *extract* tool may be used in conjunction with any other tool to provide further information on a composition's ending. For example, the *extract* and *freq* tools can be combined to provide even more statistics on the last five bars of a composition using the following format,

```
$ freq -bs b2f2p1.dms
28
$ extract -b24,28 b2f2p1.dms | freq -a
```

which counts the occurrences of each 'black' note in the last five bars (bars twenty-four to twenty-eight) of the DARMS-encoded data file 'b2f2p1.dms'. Not knowing the total number of bars in a composition is not a problem, since the *freq* tool can be employed to extract such information. The above example shows the use of the *freq* tool for such a purpose, and displays twenty-eight as the total number of bars in the data file 'b2f2p1.dms'.

Obviously, the *freq* and *endan* tools do not attempt to imitate or emulate human intuition. The computer imitation of intuition however, with a machine engaging in human-like thought processes, immediately implies artificial intelligence (AI), together with the application of programming languages such as LISP and PROLOG, which enable a computer to adopt a 'learning' process whereby if what it [the computer] produces is deemed to be a 'mistake', it will avoid producing it or making it again. However, most AI or expert systems, geared to obtaining information through learning, reasoning, justification, and

decision making, are either too simplistic for true representation, or still in a development stage. Creating a standard non self-modifying algorithm for the most frequent use of intuition, an algorithm which may be programmed via any high-level computer language, will not completely solve this intuition problem, but it will be a step towards avoiding large discrepancies in results of intuitive-based analyses.

In music analysis, intuition is most frequently exploited when attempting to conclude that two melodies are similar. In fact, the majority of analytical methods rely upon the recognition ability of a user to discover melodic repetition, transformation and variation. The development of a processable representation of this form of intuition—identification and gradation of similarity—will help procure new, efficient, effective and consistent forms of tonal, and even atonal analysis.

The Analysis Environment, therefore, requires a tool which can extract or locate melodies in a score which are 'similar' to a specified melody. The tool must also be able to calculate the degree of similarity between any two melodies. A tool of this kind requires a formula, which accounts for the shape of melodies as well as the sound of melodies in its quantification of similarity. The following pages describe the *simfind* tool, the formula used for gradation of similarity, and the method used to arrive at such a formula.

Starting simply with raw rules and later weaning them into a powerful equation, a simple formula can be built up to help calculate similarity between two series of pitches.

Initially, accounting for only the semitone distance between two compared notes, their similarity may be calculated via the following formula:

1. Assuming that the two notes are less than an octave apart,

    a.   when the distance between the two notes increases, the similarity decreases.

          Thus, where sy represents similarity, and sd represents semitone distance,

$$sy \propto 1 / sd$$

    b.   When the distance between the two notes is zero, the similarity is at its maximum, ie one. Thus,

$$sy = \frac{1}{(1 + sd)}$$

    c.   However, since the semitone distance will only play a small part in the final calculation of sy, the temporary similarity-rating achieved using sd must be decreased, and as the semitone distance may only be an integer between zero and eleven inclusive,

$$sy = 1 / \left(\frac{1 + sd}{12} \right)$$

2. For comparison of two melodies, each containing an equal number of notes, the semitone distance similarity-rating may be calculated using:

$$sy = \left(1 / \left(\frac{1 + sd(\alpha)}{12} \right) \right) \times \left(1 / \left(\frac{1 + sd(\beta)}{12} \right) \right) \times \left(1 / \left(\frac{1 + sd(\chi)}{12} \right) \right) \ldots$$

and so on, where $sd(\alpha)$, $sd(\beta)$ and $sd(\chi)$, etc, equal the semitone distance between the first, second, and third notes of the compared melodies, respectively.

3. For any octave, the frequency difference between two pitches increases by a factor of two when the two pitches are moved up an octave, indicating that pitch is exponential. A relationship such as this must be reflected in a similarity-rating calculated using only semitone distance.

Thus, taking 'middle C' as a starting pitch, the C two octaves above must have a *sy* value of half that of the C merely one octave above. Consequently,

$$sy = 1 / \left(\frac{1 + sd}{12} \right) \times \left(2^{oc} \right)$$

where *oc* equals the octave difference between two compared notes.

For example, when comparing two notes a major third apart, situated within the same octave, *sd* equals four, and *oc* equals zero; and, when comparing two notes situated two octaves and a major second apart, *sd* equals two, and *oc* equals two etc

Although a frequency difference of *x* Hz between two pitches will increase to *x* multiplied by '2^y' Hz when the same two pitches are raised by *y* octaves, it is not necessary to take this exponential curve into consideration when comparing pitches both above and below an original pitch. As exponentially-hearing humans, we can only comprehend a constant semitonal scale. Thus, regarding purely the semitone distance, both a B below and a C sharp above, have the same *sy* value when compared with middle C, even though B to C (in Hz) is less than C to C sharp (in Hz). Semitone distance *sd* is always a positive integer. For example, a minor third either above or below the original pitch has a *sd* value of three.

Quantification of similarity, however, is dependent upon more than semitone distance. Although a crude, and as yet ineffective formula for

similarity arises from this semitone difference, the degree of dissonance, for such an interval plays a more important role when defining the similarity between two notes. "...nearly all musical intervals are 'dissonances'... when passing through this continuum of dissonance, we hit from time to time upon a combination of musical sounds which... strikes us as sweet, restful, and pure; these intervals are the so-called 'consonances'... consonances have become the next-to-universal building-blocks of music".[107] In reality, any pitch sounded, other than the original pitch, may be classed as a dissonant pitch. Consequently, following in the footsteps of Helmholtz, it is possible to construct a table depicting dissonant-based similarity between compared notes.

Buck, employing partials comparison when calculating the dissonance of intervals states that "The unison and octave are free from dissonance because the harmonic chords are the same... The interval which comes next in terms of smoothness is the fifth, though in this there are obviously several dissonant partials present".[108] Enumerating common partials, generated through the simultaneous sounding of two pitches, allows the simple gradation of all intervals in terms of their dissonance (tables A and B).

Table A   Comparison of fundamental and first seven partials

Interval	Abbreviation	Common number of partials (maximum)
Unison/Octave	1 or 8	8
Minor 2nd	2-	0
Major 2nd	2	1
Minor 3rd	3-	3
Major 3rd	3	4
Perfect 4th	4	3
Diminished 5th	5-	2
Perfect 5th	5	4
Minor 6th	6-	1
Major 6th	6	3
Minor 7th	7-	4
Major 7th	7	0

---

107   Cohen, H. F., Quantifying Music, Reidel, 1984

108   Buck, P., Acoustics for Musicians, O. U. P., 1928

Table B   Order of intervals - 'partial-compared' dissonance

Least dissonant	1, 8
	5, 3, 7-
	6, 3-
	4, 5-
	2, 6-
Most dissonant	2-,7

In full agreement with Helmholtz and others, comparison of partials reveals that the minor second and major seventh "are the harshest dissonances in our scale".[109] Sadly, since an increase or decrease in compared partials produces marked variation in the calculated order of dissonance, results for other intervals are not as conclusive.

Through simple fractional mathematics and musical inversion, however, Helmholtz arranged the consonant intervals into an order of 'harmoniousness' (table C), describing the minor sixth—with its frequency ratio of 5:8 containing a number greater than five—as the most imperfect of the consonant intervals.

Table C

1.	Octave
2.	Fifth
3.	Fourth, major 3rd, major 6th
4.	Minor 3rd
5.	Minor 6th

Jeans, like Helmholtz, maintaining that "the smaller the number the better the consonance",[110] graded intervals according to the size of whole numbers used to represent frequency ratios. The Jeans method provides a more reliable table than one generated through partials comparison (table D), and as such, forms a basis for incorporating intervallic relationship in the calculation of sy. Two sets of intervals—the minor seventh and minor second, together with the major third and major sixth—occur at the same points on the

---

109   Helmholtz, Sensations of Tone, Longmans, 1877

110   Jeans, Sir James, Science and Music, Cambridge University Press, 1937

128

consonance/dissonance scale of table D; and therefore, since no two intervals can have the same consonance/dissonance value, the table needs further expansion.

Table D

Consonant		
Unison	1
Octave	2
Perfect fifth	3
Perfect fourth	4
Major third, major sixth	5,5
Minor third	6
Minor sixth	8
Major second	9
Major seventh	15
Minor second, minor seventh	16,16
Diminished fifth	27
Dissonant		

In an essay entitled "The Rationalization of a harmonically Irrational Set of Pitches",[111] Clarence Barlow described his previously created formula for calculating the 'indigestibility' of numbers. More importantly though, through elaboration upon the 'indigestibility' formula he produced a fresh algorithm for finding the harmonicity (in effect, the degree of dissonance) between two pitches within an octave.

Although Barlow's formula deems a major second to be more harmonious than both a major third and a major sixth—providing doubt as to its validity—it satisfactorily confirms that when the second (previously ignored) number of the frequency ratio is taken into account, the major sixth appears as a more harmonious or less dissonant interval than the major third; and likewise, the minor seventh appears less dissonant than the minor second.

---

111   Barlow, C., "Two Essays on Theory", Computer Music Journal, Vol. 11, Number 1, Spring 1987

Although the dissonant-graded table is now complete, its foundations—fractions of the pure Pythagorean style—are not those of the everyday tempered scale; and thus, to progress any further, a dissonant-base formula must account for the human-induced dissonance of sharpening and flattening intervals. Table E lists all intervals in their order of dissonance, along with the associated variation in frequency between real and tempered ratios.

Table E

a:	b:	c:	d:	e:	f:
1	1	1/1	1.000000000	1.000000000	0.000000000
2	8	2/1	2.000000000	2.000000000	0.000000000
3	5	3/2	1.500000000	1.498307077	0.001692923
4	4	4/3	1.333333333	1.334839854	0.001506521
5	6	5/3	1.666666667	1.681792830	0.015126163
6	3	5/4	1.250000000	1.259921050	0.009921050
7	3-	6/5	1.200000000	1.189207115	0.010792885
8	6-	8/5	1.600000000	1.587401052	0.012598948
9	2	9/8	1.125000000	1.122462048	0.002537952
10	7	15/8	1.875000000	1.887748625	0.012748625
11	7-	16/9	1.777777778	1.781797436	0.004019658
12	2-	16/15	1.066666667	1.059463094	0.007203573
13	5-	27/20	1.350000000	1.414213562	0.064213562

Key

a	Order
b	Interval name
c	Pure ratio (fraction)
d	Pure ratio (decimal)
e	Tempered ratio (decimal)
f	Variation (tempered dissonance)

A formula, therefore, for calculating dissonant-based similarity, to be used in conjunction with the formula for semitone-distance similarity, may now be written as such:

$$\alpha \times \left(\frac{1}{p} + (\beta \times v) \right)$$

where *p* is the position of an interval within the order of dissonance (ie a perfect fifth (5) has a *p* value of three), and *v* is the variation between real and tempered frequencies (ie a perfect fifth has a *v* value of 0.001692923). α and β , are variables to be defined by the user of the formula. β varies the importance of *v* within the dissonant-based formula, and α varies the importance of the dissonant-based formula itself when used in conjunction with the semitone-distance formula. Thus, the entire formula for similarity is:

$$sy = \left(\alpha \times \left(\frac{1}{p} + (\beta \times v) \right) \right) / \left(\left(1 + \frac{sd}{12} \right) \times 2^{oc} \right)$$

$$0 < \alpha <= 1 \text{ and } 0 <= \beta <= 100$$

Having established that the <u>Analysis Environment</u> is concerned only with single-line melodies, and that any arbitrary inputted 'tune' must therefore be monophonic, locating similarities within the data file becomes marginally simpler. Faced with a melodic idea, most of the present analytical systems are quite capable of employing parsing techniques to locate instances where identical melodies occur within the score. A handful of systems even allow 'wild-card' search-methods where pattern-matching succeeds even if a specified number of pitches differ between inputted and located melodies—in the <u>Analysis Environment</u>, the *all* (a - eleven) tool also has this facility. The *simfind* tool uses the above similarity formula, however, and not only has the capability of finding identical occurrences of an input-melody, but also occurrences of 'similar' melodies together with a quantification of their similarity. The *all* tool is actually a command interpreter (ie it scans and executes a list of commands). A computer program (really a list of 'Analysis 11' commands) can be executed using the *all* tool. The following command line:

```
$ all -f analysis.prg b2f2p1.dms
```

executes the 'Analysis 11' commands held in the program file 'analysis.prg'. The commands are applied to the DARMS-encoded data file 'b2f2p1.dms'. The 'Analysis 11' language enables music patterns and pattern variants to be located in a DARMS-encoded data file. A full list of commands and syntax for the language can be found in appendix B.

The *simfind* tool will compare all possible melodies, starting on any pitch within a monophonic composition, with a melody input by the user. Since this puts no undue weighting on any single pitch, a piece of ten pitches in length scanned with a two-pitch input melody will produce nine matched-melodies together with their associated similarity values. Nine two-pitch melodies create a melody of eighteen pitches in length. The eighteen-pitch melody is much longer than the original ten-pitch melody, which indicates that some of the two-pitch melodies selected by the *simfind* tool are more important than others, and warrants a technique for determining which are the more important matched-melodies.

With a DARMS-encoded monophonic composition held in memory, the *simfind* tool, having compared an inputted melody of length *x* with all *x*-length melodies in the composition—starting at the first pitch of the composition, and progressing one pitch at a time until it reaches the end—produces a list of like melodies found within the composition, together with their similarity values—ie how similar they are to the inputted melody (represented with values from 0 - not similar at all, to 1 - identical).

Assuming that during the composition of a ten-pitch melody, the composer does not conceive the overlapping of sub-melodies within the main melody, then the monophonic composition may consist of five two-pitch melodies, two five-pitch melodies, three two-pitch melodies and a four-pitch melody, or any combination of melodies totalling ten pitches in length. Therefore, for a ten-pitch monophonic composition, dealing with a list of nine

two-pitch matched-melodies (a total of eighteen pitches) is pointless, and a method has been adopted to distinguish important melodies from insignificant ones.

The eventual course of action was to select five melodies with the highest similarity (when compared with the inputted melody). This did not, however, account for the overlapping of matched-melodies, and overlapping melodies needed to be avoided. If the first five two-pitch melodies boast the five greatest similarities (diagram H), this creates overlaps on four of the six pitches covered, ignoring the last four pitches of the composition.

Diagram H.

1.   Only 5 2-pitch melodies allowed.

2.   If 1st 5 2-pitch melodies taken, the overlapping ignores the last 4 notes.

input melody
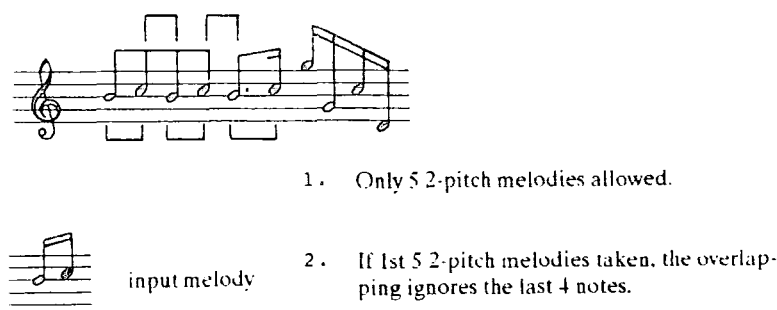
The *simfind* tool circumvents this, and similar problems by calculating which combination of two-pitch melodies would provide the highest total similarity, excluding any overlapping matched-melodies. At times, this method still leaves pitches unaccounted for, perhaps indicating that the input-melody is not a significant enough constituent element of the composition to account for all pitches.

This technique does of course create one large problem—computation time. To find the best combination of matched-melodies necessitates the examination of every feasible melodic combination. This search was carried out using a binary counting technique (diagram I), ensuring that every matched-melody was accounted for.

Diagram I.

input melody

Overlap?					0.9	0.6	0.9	0.6	0.9	0.3	0.1	0.4	0.2 similarity value	Total
No	1													0.9
No	0	1												0.6
Yes	1	1												–
No	0	0	1											0.9
No	1	0	1											1.8
Yes	0	1	1											–
Yes	1	1	1											–
No	0	0	0	1										0.6
No	1	0	0	1										1.5
No	0	1	0	1										1.2
Yes	1	1	0	1										–
Yes	0	0	1	1										–
Yes	1	0	1	1										–
Yes	0	1	1	1										–

etc.
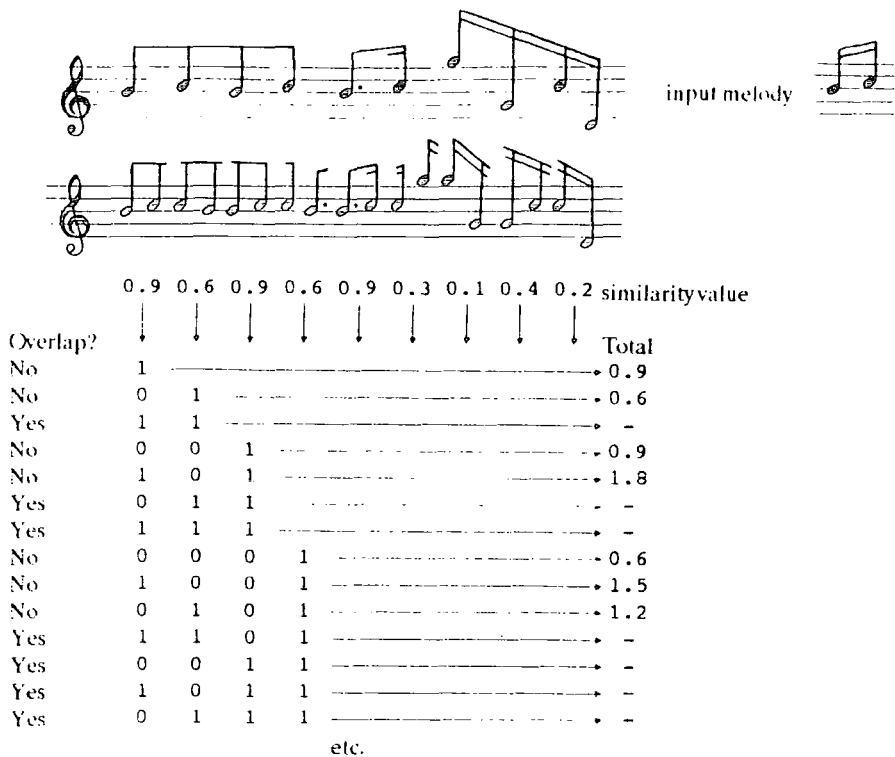
The 'on's and 'off's of each successive binary number were used as a pattern for selecting melodies from the list produced. Patterns which included overlapping melodies were ignored. Thus, the pattern with the highest similarity (ie the similarities of all the melodies, added together) became the suggested set of best matches accounting for as much as the complete composition as possible.

Unfortunately, with such a method, computation time is doubled for each successive pitch a composition contains. For example, assuming that the computation time for a one-pitch composition is only 1/1000th second, then for a two-pitch composition it becomes 2/1000ths second, and for a three-pitch composition, 4/1000ths second, and so on. Thus, for a forty-pitch monophonic composition, the computation time will be approximately seventeen years—not entirely practical for a small computer-workstation.

Since halving the computation time only permits analysis of one more pitch, even parallel-processing is not a valid alternative when considering a

134

faster machine. Therefore, for the system to be of any practical use, an alternative technique has been developed to allow fast analysis of a large number of pitches.

The new search method reduces the formidable seventeen years to a much more respectable three minutes. The method simply involves selection of the found melodies possessing the highest similarity values, but always avoiding overlaps (diagram J).

Diagram J.

0.9 0.6 0.9 0.6 0.9 0.3 0.1 0.4 0.2 similarity value

patterns with highest similarity (avoiding overlaps)   Total similarity

	Total similarity
	3.1 · -best matches
	2.2
	3.1 ←─
	2.2
	3.1 ·
	2.5
	3.1 ←─
	3.1 ··
	3.0
	3.1 ·

▭ — 2-pitch pattern selected
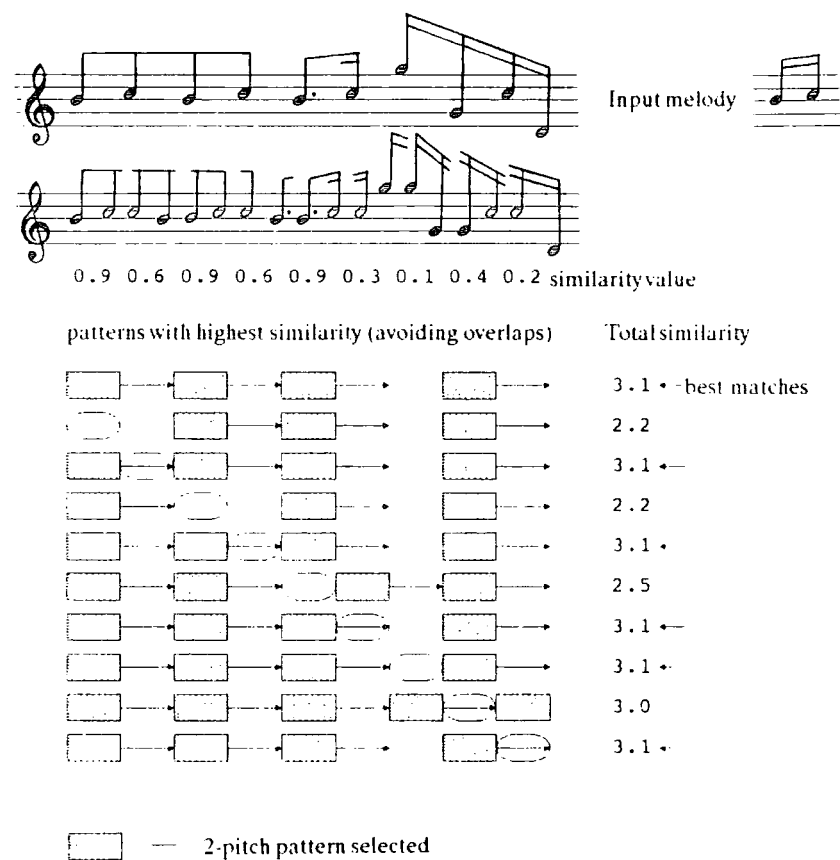
The 'eggs' in diagram J show two-pitch patterns which have been purposely discarded in the overall similarity computation. The first line calculates the highest similarity with no overlaps, the second line calculates the highest similarity with no overlaps and disregards the first two-pitch pattern, and the third line also calculates maximum similarity without overlaps but disregards the second possible two-pitch pattern, and so on.

Once achieved, a melody is deleted from the list and the selection process is repeated, each time reinstating a melody and deleting another, until all melodies have, at one time, been absent from the list. This way, all feasible patterns are accounted for, and the best pattern—producing the highest total similarity—is easily obtainable.
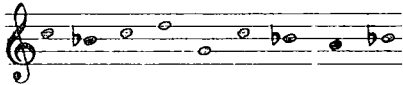
The *simfind* tool uses the following command-line syntax:

```
$ simfind -b β  -a α  -m "melody" file-name
```

where α is a value greater than zero and less than or equal to one, and β is a value greater than zero and less than or equal to one hundred. The value of α is inserted directly into the similarity formula and increases or decreases the importance of dissonance when determining similarity—the higher the value, the greater the importance. The value of β is also inserted directly into the similarity formula and increases or decreases the importance of tempered and real frequency difference, when determining similarity—the higher the value, the greater the importance. Setting α or β to zero on the command line will employ a basic similarity formula which does not use their values. By default, the output of the *simfind* tool is a list of every possible melody (the same length as the "melody" specified on the command line) together with a numerical value indicating each melody's similarity to the "melody" specified on the command line—one is maximum similarity, zero is minimum (although theoretically, a melody cannot have zero similarity with another melody). Melodies are listed in order of similarity, with the most similar at the top of the list, and the least similar at the bottom of the list. Specifying the '-c' option with the *simfind* command will list the melodies in chronological order as opposed to similarity order. Every possible melody is printed out. This means that if the "melody" specified after the '-m' parameter is five notes long, melodies consisting of notes one to five, two to six, three to seven, four to eight and so on, will be displayed together with their similarity values. This results in a tremendous
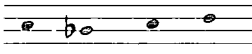
amount of superfluous data, since outputted melodies will overlap. The '-f'
option uses the 'best fit' approach to select the set of melodies which provides
the highest total similarity and accounts for the most of the DARMS-encoded
data file without any of the melodies overlapping. The '-c' and '-f' options can be
used together to produce a chronological list of the 'best fit' melodies. The
following example shows typical output from the *simfind* tool:

```
$ simfind -sc -b 0 -a 0 -m "26 25- 26 27" b2f2p1.dms
0) 26  25- 26  27   1.000000
1) 25- 26  27  23   0.397729
2) 26  27  23  26   0.453782
3) 27  23  26  25-  0.514286
4) 23  26  25- 24   0.366076
5) 26  25- 24  25-  0.600000
etc...
```
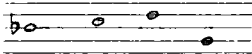

26	25-	26	27	original melody
26	25-	26	27	1.000000
25-	26	27	23	0.397729
26	27	23	26	0.453782
27	23	26	25-	0.514286
23	26	25-	24	0.366076
26	25-	24	25-	0.600000
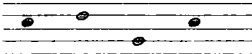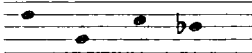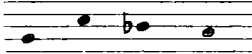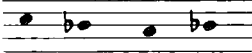
Currently there are thirty-two tools in the <u>Analysis Environment</u>. Each tool can
take a number of options to change the format of its output. Remembering the
syntax (although almost standard throughout the tools) and the significance of

each option will become more difficult when new tools are added and the scope and flexibility of the Analysis Environment is expanded. Appendix A lists the tools currently comprising the Analysis Environment, and uses the UNIX standard for documentation by placing each tool on a separate page, together with an explanation of its syntax and options. It is not, however, always possible to have a manual at one's side whilst using the computer, and consequently a *help* tool has been included within the Analysis Environment. The *help* tool, used by itself, produces a list of all the available tools within the Analysis Environment. Specifying the name of a tool on the command line, after the *help* command itself, displays the page from the manual for the specified command. The page is displayed on 'standard output' (ie the screen) and may therefore, for example, be redirected to the printer. The following example shows the output from the *help* tool when no tool name is specified:

```
$ help
Help is available on the following commands:
all        anote      bend       cursatz    darmstrip
density    endan      extract    form       freq
fsets      help       htod       intfind    invert
key        motif      play       range      rcheck
rhythm     rnote      semio      score      shape
shuffle    simfind    squash     sync       transpose
ursatz     vary
Usage: help <command>
$
```

Three more tools provide the scholar with factual information on DARMS-encoded data files. The *range* and *density* tools calculate the distance between the highest and lowest notes in a DARMS-encoded data file, and the total number of notes in a DARMS-encoded data file respectively. By default, the distance returned by the *range* tool is a 'lines and spaces' distance and does not account for sharps or flats. Since the DARMS subset used within the

Analysis Environment[112] does not use key signatures, and any accidentals (as well as the sharps and flats from the key signature) are hard-coded into the data itself, the default distance, calculated via the *range* tool, is really a 'white note' distance because all black notes are ignored. The '-c' option, however, calculates the semitone distance instead. The default output from the *density* tool may also be changed using the '-b' option. The '-b' option returns the average number of notes per bar—in essence the 'thickness' or *density* of the composition. In the following example, the *htod* tool is used to convert the Hewlett-encoded data file 'b2f2p1.hew' into subset DARMS. The subset DARMS is piped through the *density* tool, which, using the '-f' option, displays the average number of notes per bar. The subset DARMS carries on along the pipeline and is piped through the *range* tool. The *range* tool displays the range of the data, using the default 'white note' measurement. The '-s' option, used with the *range* tool, suppresses the output of the subset DARMS at the end of the pipeline.

```
$ htod b2f2p1.hew | density -b | range -s
Average bar-density: 7.586207
Range: 14 lines and spaces
$
```


highest note

lowest note   range = 14

---
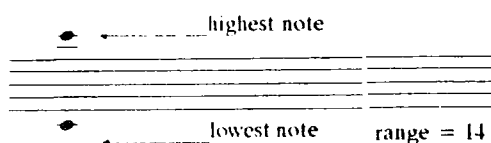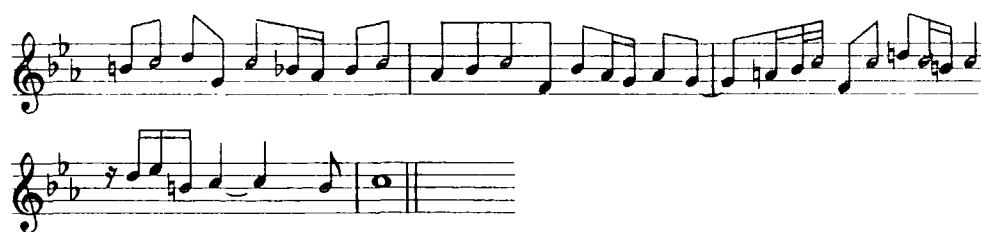
112   'Pure' DARMS data can be converted into subset DARMS (for use in the Analysis Environment) using the 'darmstrip' tool. See page 121 and page 193.

Set Theory, although it is primarily aimed at helping scholars to understand the organisation of pitches in atonal music, might well produce interesting information on some tonal music. The tonal fugues of J. S. Bach, for example, have been put together using a systematic method, just like many atonal compositions, and the scholar might glean fresh information from the scores using an atonal analytical methodology. Atonal techniques, however, disregard the harmonic implications of tonality's scale and degree. Using rests as set delimiters, the *fsets* tool displays all sets used within a DARMS-encoded data file, together with each set's location. The *fsets* tool produces three tables. The first table consists of a chronological list of sets, in 'prime' form, together with their 'normal' forms and the bar and note number on which they start. The second table is effectively a numerically sorted list of the normal forms of sets used. Ordering the sets in this fashion places the similar sets in adjacent positions. Again, the starting bar and note number of each set is displayed together with the prime form of the set (instead of the normal form), but only for those sets which have been used more than once. The third and final table is perhaps more interesting though. It consists of a chronological list of the normal forms of the sets used. This table, however, shows the relationship between the sets used—ie what pitches must be removed or added to evolve one set into another. The only options available with the *fsets* tool are the 'suppress data' and 'verbose' options ('-s' and '-v'). The tables output from the *fsets* tool have the format shown on the following page.

Musical patterns might recur and evolve throughout a composition. The more obvious variants maintain the same shape and key, and can be located using the <u>Analysis Environment</u> tools described so far. Inverted variants (ie patterns which recur backwards or upside down) and modulated variants (ie patterns which recur in different keys) are difficult to locate using standard pattern-matching techniques. To make detection of such patterns easier, the *invert* and *transpose* tools allow DARMS data to be changed before analysis.

(output from the *fsets* tool)
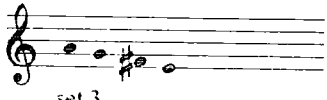
set 1	2 4 6 7 8 9 E
normal	0 2 4 5 6 7 9
set 2	1 2 4 6 7 9 E
normal	0 1 3 5 6 8 T
set 3	4 6 7 9
normal	0 2 3 5

Table 1

Prime Form	Normal Form	Bar	Note
2 4 6 7 8 9 E	0 2 4 5 6 7 9	[5,	1]
1 2 5 6 8 T	0 1 3 5 6 8 T	[7,	3]
4 6 7 9	0 2 3 5	[10,	2]

Table 2

Normal Form	Bar	Note	Prime Form
0 1 3 5 6 8 T	[7,	3]	
0 2 3 5	[10,	2]	
0 2 4 5 6 7 9	[5,	1]	

Table 3

Normal Form	Bar
0 2 4 5 6 7 9	[1]
0 1 3 5 6 8 T	[5]
0 2 3 5	[7]

Both the *invert* and *transpose* tools do not physically change the contents of DARMS-encoded data files. The result from using the *invert* and *transpose* tools is sent to 'standard output' (ie the screen) and can either be redirected to another file or through another Analysis Environment tool. The *invert* command takes a DARMS-encoded data file and inverts the data, turning it either upside

down or back-to-front. By default, the middle note of the data is used as the axis for inversion (ie when turning the melody upside down, all movement is relative to the melody's middle note). Retrogradation (ie turning the melody back-to-front) is achieved using the '-r' option and does not require an axis for inversion. When turning the melody upside down, the note used as the axis for inversion can be changed via the '-n' option. Melodies containing an even number of notes use the first of the middle two notes as the axis for inversion. The following example shows the first ten notes of the data file 'b2f2p1.dms', which have been extracted using the *extract* command, turned upside down via the *invert* tool:

```
$ extract -n1,10 b2f2p1.dms
26 25- 26 27 23 26 25- 24 25- 24-
$
```

axis for inversion

26   25-   26   27   23   26   25-   24   25-   24-
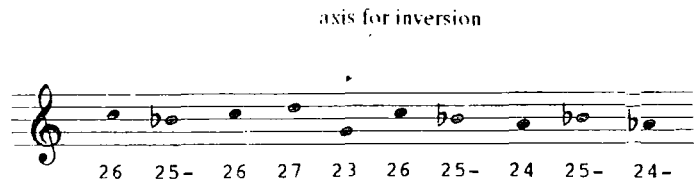
```
$ extract -n1,10 b2f2p1.dms | invert
20 21 20 19 23 20 21 22 21 22#
$
```


20   21   20   19   23   20   21   22   21   22#
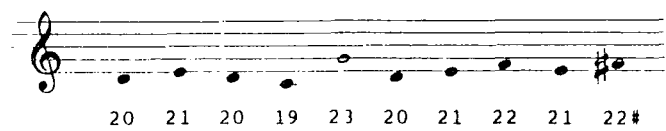
The *transpose* tool simply transposes DARMS-encoded data either up or down a specified number of semitones. The '-u' and '-d' options are used to specify the number of semitones to transpose the data up and down respectively. The following example shows the same ten notes of the previous example, which had been inverted using the *invert* tool, transposed up one semitone:
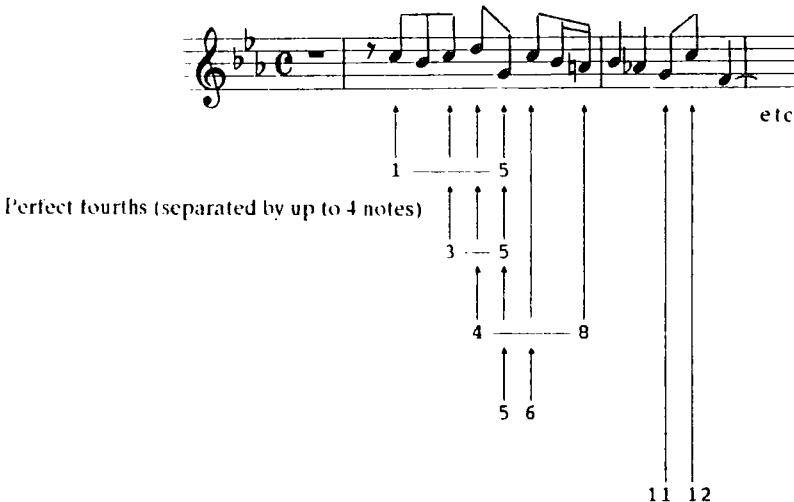
```
$ extract -n1,10 b2f2p1.dms | invert | transpose -u1
20# 22 20# 19# 23# 20# 22 22# 22 23
$
```


20# 22 20# 19# 23# 20# 22 22# 22 23

The *intfind* tool searches DARMS-encoded data files for occurrences of a specified interval. Major, minor, augmented and diminished intervals from a unison up to a fifteenth can be selected from the command line—a diminished seventh, for example, is represented by 7d, whilst a minor 3rd is represented by 3m. By default, the *intfind* tool locates exact matches of the interval. The '-N' option can be used to locate the interval with an exact number of notes separating the first and last notes of the interval, whereas the '-n' option can be used to locate the interval with any number (up to the specified maximum) of notes separating the first and last notes of the interval. The following example shows some of the output produced when searching for a perfect fourth (option '-i4') separated by up to four notes (option '-n4') in the data file 'b2f2p1.dms':

```
$ intfind -n4 -i4 b2f2p1.dms
Perfect fourth:
Start End    Direction
1      5      \
3      5      \
4      8      \
5      6      /
11     12     /
etc...
```


Perfect fourths (separated by up to 4 notes)

143

Another 'searching' tool, but one which addresses the problem of rhythm analysis as opposed to melodic analysis, is the *rhythm* tool. The *rhythm* tool, a basic rhythm pattern-matching tool, locates occurrences of a rhythmic pattern specified on the command line. The rhythmic pattern must be represented using valid DARMS rhythm symbols (eg Q for a quarter note, and E. for a dotted eighth note). If no options are selected, the *rhythm* tool locates identical occurrences of the specified rhythmic pattern. The '-d' and '-a' options locate diminished and augmented versions of the specified pattern respectively. When the lengths of notes in a rhythmic pattern are reduced, the rhythmic pattern is said to be diminished, and likewise, rhythmic patterns with increased durations are regarded as augmented. The following example locates a 'quaver, semiquaver, semiquaver' pattern in the second fugue from Bach's second book of preludes and fugues:

```
$ rhythm -s -r "E S S" b2f2p1.dms
"E S S" Start positions:
6
14
24
etc...
```


Two other tools perform rhythm analysis related functions. The *sync* tool calculates a level of syncopation based on the number of notes that sound across strong beats. By default, the *sync* tool calculates the percentage of notes which sound across the first beat of each bar. The tool examines each first beat, observing which first beats have 'struck' notes (ie notes which are played on the beat) and which first beats have 'sounding' notes (ie notes which are not played on the beat, but their duration carries the sound across the beat). The

following example calculates a percentage for the first part of the first fugue from Bach's second book of preludes and fugues:

```
$ sync -s b2f1p1.dms
struck:      59
sounding:    15
syncopation%: 20
$
```

The higher the percentage, the greater the amount of syncopation. The '-b' option can be used to specify which beats should be examined. Using the '-b3-4' option, for example, forces the *sync* tool to examine the third and fourth beats of each bar. The '-p' option displays a percentage of syncopation for each bar, calculated using data for the previous two bars and the next two bars. This means that the percentage calculated at bar ten is based on data in bars eight to twelve. The data can be graphed using the '-g' option. For example, the following command line draws a graph of progressive syncopation percentages for the first part of the first fugue from Bach's second book of preludes and fugues:

```
$ sync -pgs b2f1p1.dms
sync%
80%                                              xx
                                                xxx
                                               xxxx
                          xx                   xxxx
                        xxxxx                 xxxxx
                        xxxxxxx               xxxxx
              xxxxx   xxxxxxxxx      xxxxx    xxxxxxx
              xxxxx   xxxxxxxxx      xxxxx    xxxxxxx
0%              1           2           3           4           5
bar  12345678901234567890123456789012345678901234567890123456789012345678

sync%
80%       x
          x
        xxxx          xxx
        xxxx          xxxx
      xxxxxx         xxxxx
      xxxxxx         xxxxx
    xxxxxxxx        xxxxxxx
    xxxxxxxx        xxxxxxx
0%       6           7           8
bar  9012345678901234567890123
```

The horizontal axis depicts time, with a scale comprising bar numbers. The vertical axis shows the level of syncopation. The beginning and ending of the fugue, therefore, are deemed to contain no syncopation, whereas bars sixty to sixty-seven contain a great deal of syncopation. The *rcheck* tool examines and counts repeated pitches and durations. By default, the *rcheck* tool displays the percentage of pitch and duration repetition. The following example calculates the percentage and duration repetition for the first part of the first fugue from Bach's second book of preludes and fugues:

```
$ rcheck -s b2f1p1.dms
pitches%:    2
durations%:  42
$
```

The '-c' and '-C' options can be used to count the repetitions of each type of duration and pitch respectively. This information can be graphed using the '-g' option for repeated duration activity and the '-G' option for repeated pitch activity. Graphing the repeated duration activity of the first twenty bars in the first part of the second fugue from Bach's second book of preludes and fugues can be achieved with the following command line:

```
$ extract -b1,20 b2f2p1.dms | rcheck -Gs
```

and produces:

```
            S S
            S S
            S S
          S S S
          S S S
          S S S                                      E
      SE Q S S S          Q  ESE  Q  SE  Q  SE       E
1-2-3-4-5--6-7-8-9-10-11-12-13---14-15--16-17--18-19-20-
```
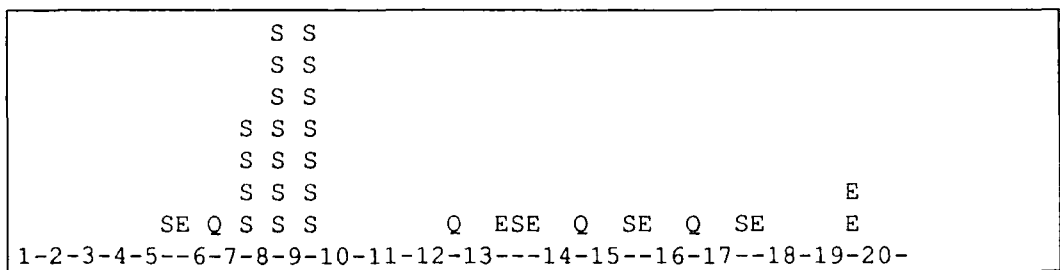
The horizontal axis represents time and shows bar numbers. The vertical axis shows the level of duration activity using DARMS notation. Bars seven to nine, for example, show a high level of semiquaver repetition.

146

The movement graph of the *endan* tool, described earlier, is based upon the *shape* tool. The *shape* tool, however, displays the overall shape of a complete DARMS-encoded data file, and not simply the last five bars. Standard characters are used to display rises and falls ('/' and '\') and the '-r' option reduces large rises and falls to single rises and falls, displaying a 'ripple' pattern ('/\/\/\'). Rests are left intact and displayed as spaces, separating the ripple patterns for each phrase. A high degree of ripples means a high degree of small up and down movement. Only a handful of ripples means large rises and falls. The '-m' option smooths out the standard graph, turning a large rise with a small fall in it into simply a large rise. The following example shows part of the default graph and part of the 'ripple' graph for the second fugue of Bach's book two:

```
$ shape -s b2f2p1.dms
1 2   3        4      5
     /\/\
    \/        \/\
               \/\
                 \
                  \/\
                    \
                     \
                      \
                         etc...
$ shape -rs b2f2p1.dms
1 2  3   4   5
   \/\/\/\/\/\
```

The *bend* tool locates the beginning and ending of melodic phrases in a DARMS-encoded data file. For the purpose of the *bend* tool, phrases are deemed to be rest-delimited melodies. By default, the beginning and ending note-number of each melodic phrase is displayed. The '-b' and '-e' options will display just the beginning and ending note-numbers respectively. A further option, the '-n' option, can be used to display the beginning and ending note-numbers for a specific melodic phrase. The following example uses the '-n'

147

option to display the beginning and ending note-numbers for the second melodic phrase in the data file 'b2f2p1.dms':

```
$ bend -s -n2 b2f2p1.dms
46 85
$
```


start and end of 2nd melodic phrase

note number 46   note number 85
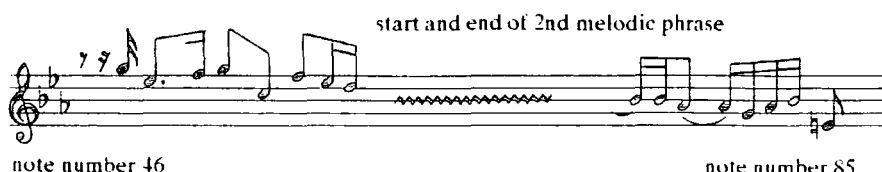
The note-number output, from the *bend* tool, may be changed to beat-number using the '-m' option. The beat-number is displayed as a multiple of the smallest rhythmic unit (ie the longest duration from which all others can be generated as integrals) in the data file. The following example uses the '-b', '-m' and '-n' options to display the starting beat-number (the 37th demisemiquaver) for the first melodic phrase in the data file 'b2f2p1.dms':

```
$ bend -sbm -n1 b2f2p1.dms
37 T
$
```


start of 1st melodic phrase

37th
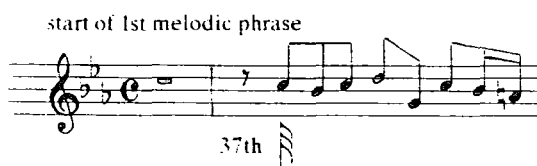
The contents then, of the 'Set One' Analysis Environment have been described—a set which comprises tools to search, extract, alter and display DARMS-encoded data. The tools of the Analysis Environment either provide structural information about the musical data, or transform the musical data in some way. A package of tools created by Camilleri et al. also carries out music analysis at a deeper level. Such computerised analysis, nevertheless, traps the programmers' analytical ideals inside the computer and blinkers the computer's results and the user's interpretations. The Camilleri team, however, did recognise the potential for combining and comparing results of different

analyses and methodologies: "we would like to emphasize the featuring points of our work... the usability of a software tool with different analytical strategies which allows the user (student, scholar or music theorist) to compare the same piece and, if feasible, to integrate them for understanding and describing the piece structure."[113]

The proof of the pudding, however, is in the eating, and although the tools can be shown to work well in laboratory conditions, the tools, and the way in which they can be bolted together and linked to the tools of both the UNIX (primarily) and MSDOS environments need to be put to the test. Chapter five, therefore, contains an analysis of the fugues (chosen primarily for their mathematical nature) of J. S. Bach's "Well Tempered Clavier" (Book II) using the Analysis Environment tools under the UNIX operating system. The challenge then, for the reader, is to make an objective decision, based on the results of said analysis, as to whether the tools and the methodology behind them and their ease of integration can benefit the scholar.

---

[113]   Camilleri, L. et al., "A Software Tool for Music Analysis", Interface, Vol 16 (1987), p.35

Testing the Tools of the Analysis Environment

The versatility of the <u>Analysis Environment</u>, through its connectivity and expandability, is not in doubt; and, the popularity of numerous toolkits currently on the market for all manner of hobbyists is proof of this. The <u>Analysis Environment</u>, likened to a 'Music-Meccano Set One' in chapter two, contains a collection of tools in its box which were placed there using an objective approach wherever possible. These individual tools, however, must be tested and their usefulness evaluated. The theory behind their integration, both with themselves and the tools of the UNIX operating system, must be put to the test and studied in practice. As such, the purpose of this chapter is to lay down a series of questions relating to a set of music data,[114] and attempt to answer such questions using the tools of the <u>Analysis Environment</u> and the UNIX environment.

Analysis of a composition can be approached in two ways. A scholar might first define a series of questions. The objective of any analysis then, would be to procure answers to the questions. Alternatively, the scholar might have no such list of specific questions, and merely expect to further his or her knowledge of the composition from close scrutiny and analysis of the score.

The trial analysis outlined in this chapter adopts both methods. The tools of the <u>Analysis Environment</u> are used to dissect or 'decompose' the scores of the fugues in an arbitrary fashion, attempting to procure new information regarding the structure of the fugues. In addition, several questions have been put forward in the hope that the <u>Analysis Environment</u> tools can aid in their solution. The intention of this chapter, once again, is to evaluate the efficiency of the tools when applied to some questions arising from an initial perusal of the

---

[114]   For the purpose of the analysis described in this chapter, the test data comprises the twenty-four fugues of Johann Sebastian Bach's "Well Tempered Clavier" Book II.

fugues. It is not, however, the intention to reinvent the wheel by producing a thorough, complete analysis and thus a textbook on how to compose fugues.

The questions themselves can be divided into two major sections—analysis of the fugue constituents (for example, the subject), and analysis of the overall fugue form (for example, the coda).

The first objective then, is to tackle questions which relate to the constituent parts of the fugue. What rules govern the labelling of a melody as 'fugue subject'? Once the subject of each fugue has been established, can a relationship between their structures, ie shape and movement, be determined? How much material of an entire fugue is based upon its subject, and is there a relationship between the subject and countersubject? How much of a fugue subject is fundamental to its structure, ie what can be removed before passing the 'breaking point'? What is the definition of elaboration, and how much can be discarded before the overall style of the subject becomes radically altered? This series of questions will form the starting point of the trial analysis. Analysis of overall fugue form will be tackled at a later juncture within the chapter.

Two approaches will be adopted when analysing the constituents of the fugue. Firstly, a single fugue, number one in C major, will be examined in great detail, and the subject will be studied in relationship to the rest of the fugue. Secondly, the subject of all the fugues will be studied in a comparative fashion to determine at what level there is a consistency of style in their form.

Since fugues are essentially contrapuntal in nature, any in-depth analysis of the first fugue should really compare the individual parts and examine the way in which they interrelate. Comparison of one melodic line with another is feasible, but taking a vertical slice from the fugue and thus a snapshot of activity of all the fugue parts at any given moment in time is not possible with the current standards imposed by the <u>Analysis Environment</u>.

Currently, only single melodic lines can be analysed. Even a single melodic line, however, should contain all the elements of style which make the melody idiosyncratic of a given composer. "...the essence of polyphonic style must be embodied in the manner in which a composer constructs an individual line.... Consequently, the style of a composer must be enshrined on the printed page in a single monophonic line, although that line would need to be of sizeable duration (eg a complete Mass movement or motet section) for meaningful deductions to be valid."[115] Despite the difficulties imposed by the Analysis Environment when attempting to undertake polyphonic or contrapuntal analysis, it is still possible to extract musical information from all parts at a specific point in time, by using the *extract* tool. The information extracted, however, cannot then be analysed using the other tools in the environment since it is essentially a vertical (chordal) DARMS structure as opposed to a horizontal (melodic) DARMS structure which is the major concept behind the Analysis Environment, ie 'piping' melodies through tools. Under such circumstances, one would have to resort to manual analysis.

Initially the data must be collected, ready for fugue-subject analysis. In simplistic terms, the fugue subject can be regarded as the initial melody of the whole fugue. Thus, the first note of the fugue is the first note of the fugue subject, and as one might expect, all of the fugues begin with a single melody— the fugue subject itself. Such an apparently simplistic deduction can be confirmed using the *bend* tool (which displays the beginning and ending positions of all melodic phrases). When examining the first fugue, the following four commands will display the starting points, within the fugue, of the fugue parts:

---

115   Morehen, J., "Statistics in the Analysis of Musical Style, Proceedings of the Second
      International Symposium on Computers and Musicology, Orsay, 1981, (Paris, CRNS,
      1983), p. 171

```
$ bend -sbm -nl b2f1p1.dms
35 S
$ bend -sbm -nl b2f1p2.dms
3 S
$ bend -sbm -nl b2f1p3.dms
67 S
$
```

and shows that part two (stored in the data file 'b2f1p2.dms') begins before all

the other parts, at the third (3) semiquaver (S) position. Producing similar

information for all the fugues requires the *bend* tool to be enclosed in a small

UNIX shellscript[116] loop, which will repeat the same commands for each fugue.

The alternative to a small shellscript loop would be to type in over eighty-one

individual commands—a task prone to typing errors and which would defeat

entirely the object of using the computer as an efficient time-saving aid. The

loop may also form the basis for any comparative testing since it can be tailored

to apply the same series of commands to all fugues. The convention adopted

for naming fugue data files allows a shellscript loop to apply commands to all

fugues data files using the following syntax:

```
for datafile in `ls b2f*p*.dms`
do
        ANALYSIS COMMANDS
done
```

The UNIX command 'ls' produces a list of data-file names stored on the

computer. The pattern 'b2f*p*.dms' limits the list produced by the 'ls' command

to those data-files which are named using the convention outlined earlier in this

thesis, ie b2 - book two, f - fugue, * - any number (UNIX), p - part, * - any

number (UNIX), .dms - suffix indicating a DARMS-encoded data file. The list

produced by the 'ls' command, therefore, comprises the names of the data files

containing all the parts of all the fugues. The shellscript 'for' loop is a text-based

---

116   See appendix C for a brief outline of the UNIX shellscript programming language.

loop and not a numeric-based loop. This means that each of the items in the list

produced by the 'ls' command is put in turn into the shellscript variable 'datafile'

and is available within the loop. The commands in between the 'do' and 'done'

will be applied to the data-file name currently stored in the shellscript variable

'datafile'. A loop, therefore, to apply the *bend* tool to all parts of all fugues would

use the following syntax:

```
for datafile in `ls b2f*p*.dms`
do
        bend -sbm -nl $datafile
done
```

where the '$' symbol in front of the variable 'datafile' is used to procure the

contents of the variable, ie the actual name of the DARMS-encoded data file.

The small shellscript program in its current format, however, simply

produces a stream of numbers, ie the beginning position of the first melodic

phrase of each DARMS-encoded data file. Inserting the UNIX 'echo' command

(which just echoes a string of text onto standard output, ie the screen) into the

middle of the loop will annotate the output, and the following use of the 'echo'

command displays the contents of the variable 'datafile' (the actual name of the

data file), and suppresses the carriage return (achieved by inserting \c into the

text string) to display the beginning position (of the first melodic phrase)

immediately after the actual data-file name:

```
for datafile in 'ls b2f*p*.dms'
do
        echo "$datafile \c"
        bend -sbm -nl $datafile
done
```

The small shellscript loop produces the following output:

```
b2f1p1.dms  35 S
b2f1p2.dms  3 S
b2f1p3.dms  67 S
b2f2p1.dms  37 T
b2f2p2.dms  5 T
b2f2p3.dms  101 T
b2f2p4.dms  585 T
b2f3p1.dms  21 T
etc...
```

which requires further 'neatening' to become more readable and comprehensible.

An alternative form of loop, which enables a series of commands to be applied to each fugue, could have the following syntax:

```
for fuguenumber in 1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24
do
        ANALYSIS COMMANDS
done
```

which puts the numbers 1 to 24 in turn into the variable 'fuguenumber' and allows the variable to be used within the 'do' and 'done' section. A 'nested' loop, ie another 'for' loop inside the 'do' and 'done' section of the previous loop, enables commands within the inner 'do' and 'done' section to be applied to the individual parts of each fugue. For example, the following shellscript:

```
for fuguenumber in 1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24
do
        for partnumber in 1 2 3 4
        do
                bend -sbm -nl "b2f${fuguenumber}p$partnumber"
        done
done
```

also applies the *bend* tool to each part of every fugue. This shellscript, however, because it uses a separate inner loop for each fugue, can be annotated using 'echo' commands as follows to produce a more readable form of output.

(Brackets have been used around the variable 'fuguenumber' in the inner loop to separate the variable name from the letter 'p' and avoid confusing the UNIX shellscript interpreter which might inadvertently look for the contents of the variable 'fuguenumberp' as opposed to the variable 'fuguenumber'.)

```
for fuguenumber in 1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24
do
        echo "Fugue number: $fuguenumber"
        echo "Parts begin at..."

        for partnumber in 1 2 3 4
        do
                bend -sbm -n1 "b2f${fuguenumber}p$partnumber"
        done

done
```

The above shellscript program, involving a nested loop, produces the following output:

```
Fugue number: 1
Parts begin at...
35 T
3 T
67 T
Fugue number: 2
parts begin at...
37 T
5 T
101 T
585 T
Fugue number: 3
parts begin at...
321 T
5 T
7 T
Fugue number: 4
parts begin at...
19 S
49 S
1 S
Fugue number: 5
parts begin at...
67 S
27 S
1 S
etc...
```

156

The above data was intended to determine which part began each fugue. Such statistics, however, provide interesting information by themselves. The above output can be arranged into a tabular format (Table A).

Table A						
1	S	35	3	67		C Major
2	T	37	5	101	585	C Minor
3	T	21	5	7		C# Major
4	S	19	49	1		C# Minor
5	S	67	27	3	83	D Major
6	S	33	1	81		D Minor
7	S	321	209	97	1	Eb Major
8	S	131	3	35	99	D# Minor
9	E	73	49	25	1	E Major
10	S	3	97	195		E Minor
11	T	7	55	157		F Major
12	S	1	33	89		F Minor
13	S	1	65	129		F# Major
14	S	55	7	127		F# Minor
15	T	3	87	171		G Major
16	S	101	53	5	149	G Minor
17	T	69	5	165	229	Ab Major
18	S	1	49	145		G# Minor
19	S	67	27	3		A Major
20	T	169	73	9		A Minor
21	E	26	2	74		Bb Major
22	E	49	1	193	121	Bb Minor
23	S	209	145	65	1	B Major
24	T	73	1	181		B Minor

The table lists the part entry-positions for all the fugues, and is divided into seven columns. The first and last columns contain the fugues number and fugue key, respectively. The second column shows the smallest duration in the fugue (DARMS notation) used by the *bend* tool to calculate the entry position of the fugue parts. Columns three to six contain the actual entry positions for each part of the fugue (as a multiple of the value in column two). If there are only three parts in a fugue, the sixth column is blank.

Ideally, in a paradigmatic sense, each part will begin after a consistent length of time. Fugue nine in E major, for example, is a classic case. The fugue contains four parts. The lowest part enters first on beat one, and the rest of the parts enter in ascending order, each twenty-four eighth (E) beats after the preceding part.

Fugue IX, E major
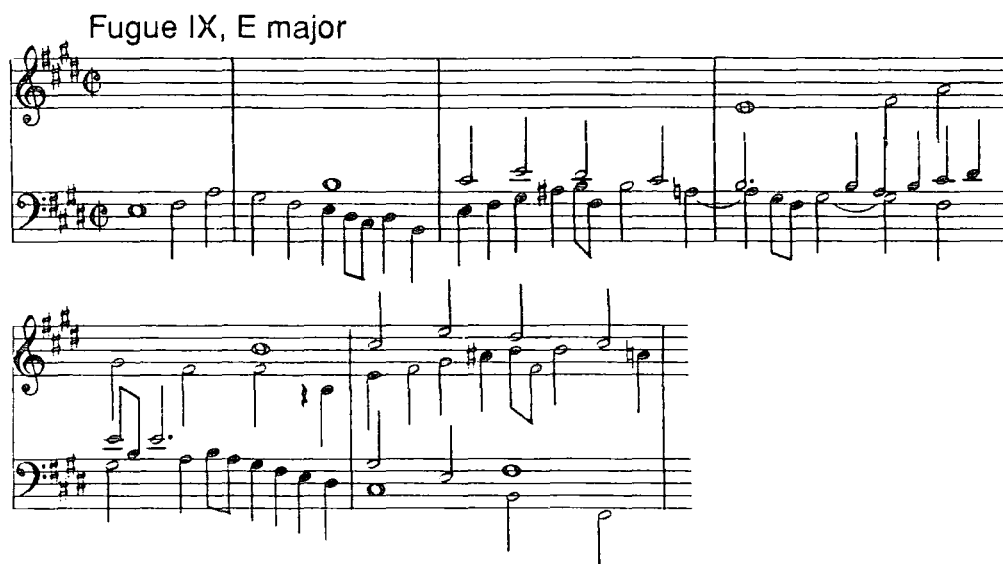
In other fugues, such as fugue sixteen in G minor, parts begin after a consistent length of time, but in an inconsistent order. The parts of fugue sixteen enter in the order of third, second, first and fourth (where the first part is the topmost part). The length of time between each part-beginning is forty-eight sixteenth (S) beats.

Fugue XVI, G minor

For some four-part fugues, which do not have each part entering after a consistent length of time, the length of time between the first and second entries is the same as the length of time between the penultimate and last entry.

Fugue VIII, D# minor

The majority of fugue parts, however, enter in an inconsistent order, and after an inconsistent time.

Two fugues stand out in table A. The first, fugue two in C minor, is unusual because the last part does not enter until the fugue is halfway through, whereas most fugues have all parts under way before the tenth bar.

Fugue twenty-one in Bb major stands out because its part entry numbers are odd, whereas the part entry numbers for other fugues are even. In fugue twenty-one there are no subdivisions of the standard duration (quaver) except six bars before the end.

Fugue XXI, Bb major
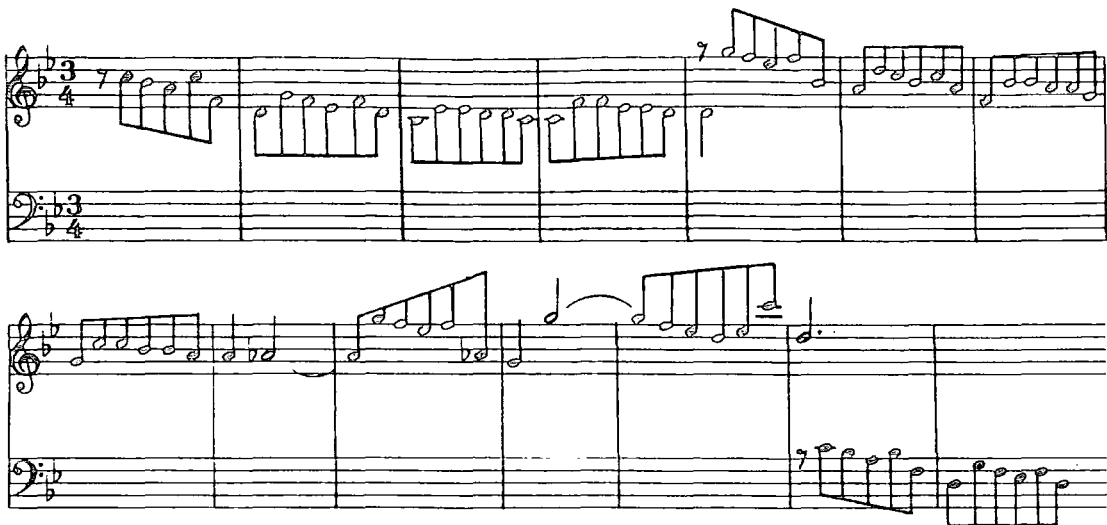
Many might argue that the data in Table A is no more informative than the scores of the fugues themselves. Indeed, there is certainly no extra information in the table, which is not present in the scores. However, the table sets out the known data using another format, and places emphasis on new areas. Infrequent rhythmic symbols stand out, and high, low and sequential number series draw attention themselves, begging for further questions to be asked.

Officially, then, the fugue subject begins with the first note of the fugue itself. As a convenient criterion for computing, the fugue subject can be taken as ending when another part enters, although from a musical point of view this does not always make sense. For example, in fugue three there are overlaps of seven quavers between the first and second entries alone. With this information, the *bend* tool can be used to find out when the second entry of a fugue begins and thus when the fugue subject ends. Having established the beginning and ending points of the fugue subject, the *extract* tool may be used to extract the fugue subject, whereupon it can be redirected into another data file, ready for use at a later date. When examining the first fugue, the following shellscript loop establishes the earliest two entry points of the fugue parts:

```
for part in 1 2 3
do
        entrypoint=`bend -sbm -nl b2flp$part`
        entrypoints="$entrypoints\n$entrypoint"
done

start=`echo $entrypoints | sort -n | sed -n "1p"`
end=`echo $entrypoints | sort -n | sed -n "2p"`

for part in 1 2 3
do
        entrypoint=`bend -sbm -nl b2flp$part`
        if [ "$entrypoint" -eq "$start" ]
        then
                partname=$part
        fi
done
```

whilst the following use of the *extract* tool may be used to put the fugue subject into a data file called 'b2f1s.dms' and adheres to the convention used for the data-file names for data files containing fugue subjects:

```
extract -t$start,$end $partname > b2f1s.dms
```

In the above shellscript program, the first 'for' loop uses the *bend* tool to calculate the starting point of each part. These starting points are separated by newlines and assigned to the variable 'entrypoints'. Outside the loop, the entry points (now stored in the variable 'entrypoints') are piped through the UNIX 'sort' tool to sort them into numerical order. The sorted list is then piped through the UNIX 'sed' (stream editor) tool which extracts the first line (the lowest number, and thus the earliest entry point and the start of the fugue subject) and assigns it to the variable 'start'. The second line of the sorted list (the second lowest number, and thus the second entry point and the end of the fugue subject) is extracted and assigned to the variable 'end'. The second 'for' loop examines each part in turn, in an effort to establish which part had the lowest entry point, and thus contains the fugue subject. The name of the part containing the fugue subject is assigned to the variable 'partname'.

161

A larger shellscript program may be built up to transfer all fugue subjects to data files called b2f*x*s.dms, where *x* represents the fugue number.

From the shellscript outlined so far, the fugue subject from the first fugue is as follows:

```
R 23 22 23 19 / 24 23 / R 22 21 22 23 21 22 /
20 22 21 22 23 24 22 23 / 21 20
```
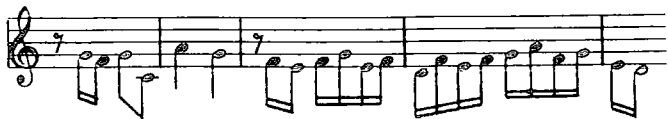

and the fugue subjects for the other fugues are as follows:

```
1.    R 23 22 23 19 / 24 23 / R 22 21 22 23 21 22 /
      20 22 21 22 23 24 22 23 / 21 20
```
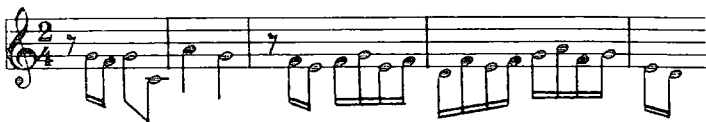


```
2.    R 23 21- 22 23 19 22 21- 20 / 21-
```



```
3.    R 12# 14# 12# 16#
```
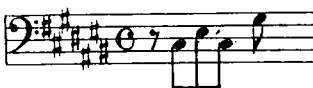
4.   12# 11# 12# 13# 12# 13# 9# 10# 11# 12# 13# 14 / 15# 14 15#
       13# 16# 15#

5.   R 20 20 20 16 18 / 18 14 17 16 15#
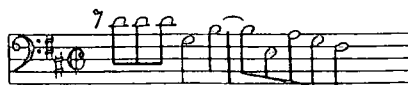
6.   20 21 22 23 22 21 22 23 24 25- 24 23 24 27 26# 26 /
       25 25- 24 23 22 21 24

7.   14- / 18- R 17- / 16 19 18- / 17- 17- 16 17- 19 /
       15 18- 17- / 16 16 15 16 18-
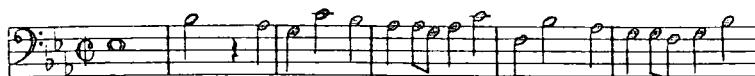
8.   R 20# 20# 20# 19## 20# 21# / 21# 20# 23# 22# 21# 24# 23# /
       22#

9.   14 15# 17 / 16# 15#
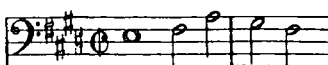
10.   R 21 22# / 23 24 23 22# 23 24 25 24 23 24 / 25 23 21 26 /
       26 25 27# 28 24 / 24 23 27# 28 22# / 22# 23 24 23 22# 21
       20# 26 25 24 23 22# / 23 24 25 24 23 22# 21 28 27 26#

11.   R 22 21 22 / 26 R 24 23 24 / 27 R 26 27 28 / 29 28 27 26
       27 25- / 24 23 22

12.   R 26 / 22 22 22 27- 25- / 21 21 21 22 23 24- / 25- 24-
       23 24- 25- 27- 26 25- / 24- 23 22 23 24- 25-

13.   R 21# 20# 21# / 22# R 19# 20# / 21 20# 19# 18 17# 18 19# /
       20# 21# 22# 20# 23# 18 / 18 17# 16#

14.   R 19# 17 15# 20 / 20 19# 18 19# 22# 18 / 18 17 16# 17 15#
       16# / 15# 16# 17
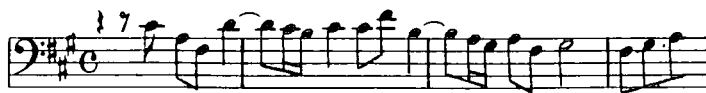
15.　　R 27 25 27 30 27 / 25 27 23 27 25 27 / 28 26 24 26 22# 24
　　　　/ 27 25 23 25 21 23 / 26 24 22# 24 20 22# / 25 24 23 24 25
　　　　26 / 27 24 25 26# 27 28 / 29#

16.　　R 20 R 18- / 21- 19 R 17 / 20 18- R 16 / 19 19 19 19 19 19
　　　　/ 19 18- 17

17.　　R 28- 26 29 25- 26 27- 28- 20- / 24- 25- 26 27- 28- 26
　　　　27- 25- 26 27- 28- / 26 25-

18.　　23# 24# 25 24# 27# 20# / 23# 25 24# 25 24# 23# / 24# 25
　　　　26# 25 28 23# / 24# 26# 25 26# 25 24#

19.　　R 17 18 19# 18 17 19# 18 20 19# 21 20 / 20 22# 21 20 21 18
　　　　19# 20 19#

```
20.    R 21 19 22 / 16# R 20 18 21 / 19 17
```
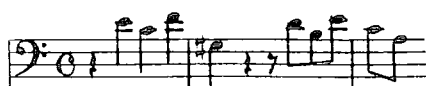


```
21.    R 26 25- 24 25- 22 / 20 23 22 21- 22 20 / 18- 21- 21- 20
       20 19 / 19 22 22 21- 21- 20 / 20 21-
```



```
22.    18- 19 R 20- / 21- 17 18- R 19 / 20- 18- 19 20- 21- 19
       20- 21- 22 20- 21- 22 / 23- 21- 22 R 23- 19 20- 21-
```



```
23.    11 13# / 16# 14 / 12# 17# / 18 17# 16# 15# 14
```
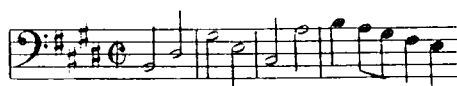


```
24.    R 22# / 20 18 17# / 18 19# 20 21 22# / 23 16 23 / 22# 15#
       22# / 21 22# 21 20 19# / 20 22# 21 20
```


The fugue subject itself is fundamental to the composition. Using the *rnote* and *anote* tools, it is possible to remove repeated and auxiliary notes a few at a time until the subject loses its 'identity', ie no longer bears any meaningful resemblance to the original. One tool in the Analysis Environment is designed specifically for quantifying similarity (*simfind*) and could be useful for monitoring

the gradual decrease in similarity as repeated and auxiliary notes are removed. This in itself would be a worthwhile exercise to establish a value for the 'breaking point', where similarity values less than the breaking point show melodies which bear no resemblance to the original, and similarity values greater than the breaking point show melodies which appear similar in some form to the original. The disadvantage with the current version of the *simfind* tool is that compared melodies must contain the same number of pitches.

Tools which can be used to monitor the disintegration of the subject are the *shape* tool (which displays the overall shape of a melody), the *score* tool (which displays a melody in pseudo-music notation) and the *play* tool (which plays a melody using the built-in speaker of a personal computer and is currently only available in the more restricted MSDOS environment). Since the rhythm of the subject will doubtless be altered radically if notes are removed, it is sensible to make only melodic comparisons of the melodies, and the *play* tool should be used without the '-r' option, enabling the *play* tool to play the melody using the same duration for every note. The *score* and *shape* tools have no facility for displaying durations.

A small shellscript loop, to display the fugue subject as it is decomposed, might have the following syntax:

```
original=`cat b2f1s.dms`
echo $original
decomposed=`echo $original | rnote -r | anote -r`
echo $decomposed
while [ "$original" != "$decomposed" ]
do
        original=$decomposed
        decomposed=`echo $original | rnote -r | anote -r`
        echo $decomposed
done
```

The program itself displays the subject in its original subset DARMS notation. The first line uses the UNIX 'cat' command to send the contents of the fugue-subject data-file to standard output, ie the screen, and assigns the standard

167

output to the variable 'original'. At this point, the variable 'original' contains the fugue subject from the first fugue, and may now be used at any point within the rest of the program. Line number two displays this original fugue subject, whilst line three pipes the original fugue subject through the *rnote* and *anote* tools using their recursive ('-r') options (which guarantee the global removal of repeated and auxiliary notes). This decomposed version of the fugue subject is assigned to the variable 'decomposed' and is displayed on the screen in the next line of shellscript. The body of the shellscript program is a 'while' loop. Unlike the 'for' loop which progresses through a list of text items, and terminates when there are no text items left, the 'while' loop will execute all commands within the 'do' and 'done' whilst a certain condition is true. The condition itself appears inside square brackets, after the 'while' command itself. This particular condition states that all commands inside the 'do' and 'done' should be executed whilst the contents of the variable 'original' is not the same as the contents of the variable 'decomposed'. In English, this means that the body of the loop will be performed until the decomposition process produces a melody which is the same as the previous melody, ie no further decomposition has taken place. Inside the loop, the decomposed melody becomes the new original melody, the original is decomposed a stage further and this freshly decomposed melody is printed out. The whole shellscript program, when executed, produces a list of DARMS-encoded melodies with the original at the top, and the most decomposed at the bottom.

Any comparison of melodies by the scholar is difficult when faced with an unfamiliar notation such as DARMS. The *score* tool, however, may be bolted onto the end of the 'echo' commands to display the melodies not in DARMS, but in the pseudo music notation.

```
echo $original | score
```

Likewise, the shape of the melodies may be displayed by bolting the *shape* tool onto the end of the 'echo' commands,

```
echo $original | shape -s
```

and both shape and notation may be displayed by bolting both tools onto the 'echo' commands.

```
echo $original | shape | score
```

The *play* tool, however, since it has no equivalent in the UNIX environment, must be used from within MSDOS. Unfortunately, although MSDOS has a restricted version of the UNIX pipe, filter and redirection facilities, it has no equivalent of the powerful UNIX shellscript programming language. All that is available is a rather terse batch programming language. Despite this, it is still possible to craft a small program which will play the melodies as they are decomposed. Such a program could have the following format:

```
            play b2fls.dms
            rnote -r b2fls.dms | anote -r > tempdata
:loopstart
            play tempdata
            rnote -r tempdata | anote -r > tempdata
            goto loopstart
```

Although somewhat more succinct that the previous shellscript program, this MSDOS batch program has no method of ending and must be aborted by pressing the computer's interrupt key sequence when the decomposition process no longer alters the state of the melody. Since there are no variables in the MSDOS batch language, the intermediate decomposed melodies have to be stored in the data file 'testdata' via redirection, and used later in the program. The goto command forces the computer to go back to the line labelled 'loopstart'. So that the scholar can monitor the decomposition process

visually as well as audibly, the '-s' option has been omitted from the *play* tool, which displays the DARMS data on standard output, ie the screen.

Fugue I, C major (during the decomposition process)

```
R 23 22 23 19 / 24 23 / R 22 21 22 23 21 22 / 20 22 21 22 23 24
22 23 / 21 20
```



```
R 23 23 19 / 24 23 / R 22 22 23 21 22 / 20 22 22 23 24 22 23 /
21 20
```
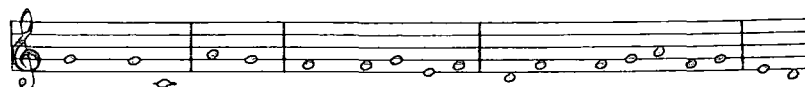


```
R 23 19 / 24 23 / R 23 21 22 / 20 22 23 24 22 23 / 21 20
```
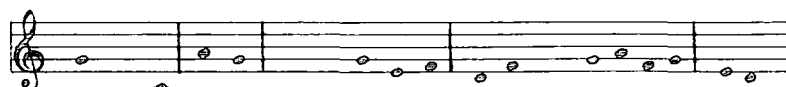


```
R 23 19 / 24 23 / R 21 22 / 20 22 23 24 22 23 / 21 20
```


The above example shows the subject of the first fugue as it is decomposed. The pictorial representation of the decomposition process does not, however, convey the same information as the audible representation by the *play* tool. Generally, when the subjects are piped through the *play* tool during decomposition, the melodies sound disjointed and unlike their originals. At the end of the decomposition process, however, the melodies are more

scalic and although somewhat more condensed, have a similar audible style to their originals. The view that auxiliary notes and repeated notes are merely decoration and not fundamental to a melody's structure would, therefore, seem to be valid.

If there is a breaking point, and a fugue subject can be condensed, the condensed version can be used successfully in further analysis, and any deductions made on the condensed version apply equally to the original version. The subject is the most important element of the fugue. Not only is it the first melody of the whole composition, but it has the honour of being rendered by each part at least once. Identical repetition of the fugue subject within a part can be discovered using the *motif* or *simfind* tools.

When using the *simfind* tool, the entire fugue part will be searched thoroughly and every possible melody of the same length as an inputted melody will be compared and a value of similarity will be displayed. Searching the second fugue part, from the first fugue, for material based on the fugue subject would be achieved using the following shellscript,

```
simfind -s -b0 -a0 -m
"23 22 23 19 24 23 22 21 22 23 21 22 20 22 21 22 23 24 22 23 21
20" b2f1p2.dms
```

if the DARMS notation is known for the fugue subject. Alternatively, the data file 'b2f1s.dms' contains the fugue subject, and can be incorporated into the above shellscript using the following syntax:

```
simfind -s -b0 -a0 -m "`cat b2f1s.dms`" b2f1p2.dms
```

The output generated is a list of melodies in order of similarity—the most similar first, and the least similar last. This list of numbers draws attention to melodies of importance, although the restriction to melodies of identical length to the

fugue subject is somewhat limiting. The rest of the parts may be checked at the same time by enclosing the *simfind* command in another 'for' loop.

```
for part in 1 2 3
do
        echo "Part: $part"
        simfind -s -b0 -a0 "`cat b2f1s.dms`" b2f1p$part.dms
done
```

The *motif* tool enables music material to be searched for a specified melody. Although only identical matches are displayed, using the '-n' option will allow found melodies to be longer than the original melody. The *motif* tool can be used to search the fugue parts for melodies which might have 'evolved' from the fugue subject.

The subject of the first fugue is twenty-two notes long. If the *motif* tool is used in the following fashion,

```
motif -sn27 -m "23 22 23 19 24 23 22 21 22 23 21 22 20 22 21 22
23 24 22 23 21 20"
```

the figure '27' after the '-n' option allows the located melodic material (which would otherwise be an identical match of the fugue subject) to be an expansion of the fugue subject by up to five notes. This expansion allows for the addition of extra repeated or scalic passing notes within repetition of the subject material.

The *semio* tool, used with '-n22' will look for identical occurrences of twenty-two note melodies. The *semio* tool, however, examines the DARMS-encoded data file in minute detail, picking up each twenty-two note melody in turn and displaying the positions of its other occurrences within the score. The first twenty-two note melody used for its search is the fugue subject itself, and its locations, displayed using the *simfind* tool, are confirmed by the *semio* tool. The rest of the output from the *semio* tool, however, shows the frequency of use

of other twenty-two note melodies, and this will draw attention to melodies of

importance and interest. Frequently used melodies must be of some structural

significance. Twenty-two notes is actually a lot of melody, and more overall

structural information might be gleaned if the melodic units being examined

were of a smaller size. If the *semio* tool is again enclosed in a small shellscript

loop, melodic units of sizes from the smallest (a single note) to the largest (a

melody the length of the entire fugue) could be examined and structural

material which is repeated (no matter what the size) would be highlighted and

be available for further close scrutiny. Such a shellscript program, which uses

the *density* tool to calculate the total number of notes in the first part of the fugue

and thus determine the length of the largest melodic unit, might have the

following format:

```
maximum=`density -s b2f1p2.dms | awk '{print $NF}'`
counter=1
while [ "$counter" -le "$maximum" ]
do
        echo "Melodic units of length $counter"
        semio -sn$counter b2f1p2.dms
        counter=`expr $counter + 1`
done
```

The first line of the above shellscript program uses the default output of the

density tool to assign the total number of notes in the data file 'b2f1p2.dms' (ie

the second part of the first fugue) to the variable 'maximum'. The output from

the *density* tool consists of the words 'Piece density:' followed by the density

itself. Piping the output from the *density* tool through the UNIX 'awk' tool (which

is yet another programming language in its own right) extracts the last field of

the output, ie the actual density value and not the text. The density, or total

number of notes, is then available via the variable 'maximum' to the rest of the

program. The variable 'counter' contains the current size of the melodic unit to

be used with the *semio* tool, and is assigned the smallest size of one, on the

second line of the shellscript program. The 'guts' of the program (a 'while' loop)

terminates when the contents of the variable 'counter' is no longer less than or

equal to the contents of the variable 'maximum', ie the melodic-unit size is greater than the fugue-part length. Inside the loop, an 'echo' command displays the current size of the melodic unit, and the *semio* tool uses the current size of the melodic unit on the data file 'b2f1p2.dms'.

Since the same data-file name occurs in two separate instances within the program, it too could be assigned to a variable and the variable used instead of the actual data-file name. This would make the program more flexible, and, used in conjunction with the UNIX 'read' command could be made to prompt the user for the name of a DARMS-encoded data file. Such an amended program might appear as follows:

```
echo "Enter name of DARMS file: \c"
read datafile
maximum=`density -s $datafile | awk '{print $NF}'`
counter=1
while [ "$counter" -le "$maximum" ]
do
        echo "Melodic units of length $counter"
        semio -sn$counter $datafile
        counter=`expr $counter + 1`
done
```

The initial 'echo' command provides a prompt for the user, whilst the 'read' command simply assigns what the user types on the keyboard into the variable 'datafile'. UNIX shellscript is a text-based language, and as such, offers no facility for mathematics or calculations of any form. Fortunately, there are several tools available under UNIX for performing mathematical calculations. The 'expr' tool is perhaps the most simple, and can be used to perform integer mathematics involving addition, subtraction, multiplication and division. The last line in the loop, therefore, uses the 'expr' tool to increase the melodic-unit size by one and assign it to the variable 'counter'. As one might imagine, output produced by the *semio* tool for melodic units of size one to three hundred and sixty-seven (which is the total number of notes in the second part of the first fugue) takes up more than one or two sheets of paper. Rather than reproduce

all of the output in this chapter, a summary of the important information is
shown below:

Fugue I, C major (part two, twenty-two note melodies)

```
----------------------------------------------------------------
3   1   3   8   5   3   1   0   1   3   0   1   10  1   0
^1  ^2  ^3  ^4  ^5  ^6  ^7  ^8  ^9  ^10 ^11 ^12 ^13 ^14 ^15
----------------------------------------------------------------
1   2   3   4   5   6   7   8   9   10  11  12  13  14  15
243 244 245 246 247 248 249 250 251 252 253 254 255 256 257

----------------------------------------------------------------
1   3   5   1   3   0   10  0   1   0   1   10  0   8   7
^16 ^17 ^18 ^19 ^20 ^21 ^22 ^23 ^24 ^25 ^26 ^27 ^28 ^29 ^30
----------------------------------------------------------------
16  17  18  19  20  21  22  23  24  25  26  27  28  29  30
258 259 260 261 262 263 264
```

Using the *semio* tool on melodies of twenty-two notes (in the second part)
immediately shows that there is repetition of the fugue subject at note number
243, ie bars forty-seven to fifty-one. Such information is only obtainable from
the score by close and careful examination.
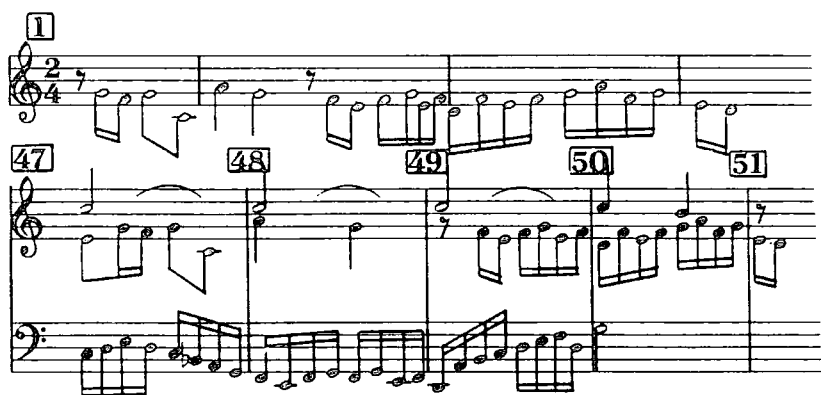
Fugue I, C major (part two, four note melodies)

```
--------------------------------------------------------------
3   1   3   8   5   3   1   0   1   3   0   1   10  1   0
^1  ^2  ^3  ^4  ^5  ^6  ^7  ^8  ^9  ^10 ^11 ^12 ^13 ^14 ^15
--------------------------------------------------------------
1   2   3   4   5   6   7   8   9   10  11  12  13  7   15
243 244 245 246 247 248 14  192 112 49  253 254 255 14  257
290 291 292 293 336 337 108 250 171 113             108
332 333 334 335         167     193 172             167
                        191     251 194             191
                        249         252             249
                        256                         256
```

Using the *semio* tool with four-note melodies produces the above information, showing that the opening fugue subject is also repeated (in a shorter form) at notes 290 and 332, ie bars fifty-six and seventy-two.

Fugue subject material might well appear in a transposed or even inverted format. The previous shellscript programs may be adapted to search for fugue material which appears in a transposed or inverted form by using the *invert* and *transpose* tools on the fugue subject before any searching takes place. The following short shellscript program raises the fugue subject by a semitone at a time (up to a maximum of thirteen semitones), effectively

transposing it into every key, and uses each transposed version with the *simfind* tool, within the 'while' loop:

```
distance=0
while [ "$distance" -le 13 ]
do
        melody=`transpose -u$distance b2fls.dms`
        echo "Distance transposed: $distance"
        simfind -s -b0 -a0 "$melody" b2flp2.dms
        distance=`expr $distance + 1`
done
```

Searching for inverted fugue subjects can be achieved using a similar method, except that the DARMS-encoded data file 'b2f1s.dms' should be inverted before being assigned into the 'melody' variable as opposed to being transposed:

```
melody=`invert b2fls.dms`
```

By default, melodies are inverted vertically (ie turned upside down) and the axis used for inversion is the middle note. If one wishes to carry out a comprehensive search for possible use of fugue material, the direction of inversion should be changed, and the axis for inversion should be tried as each note of the melody. Not only this, but the various inverted melodies should be transposed into each key before searching as well. To accomplish this, the axis for inversion can be moved along the fugue subject using the following shellscript program:

```
length=`density -s b2fls.dms`
counter=1
while [ "$counter" -le "$length" ]
do
        melody1=`invert -n$counter b2fls.dms`
        melody2=`invert -rn$counter b2fls.dms`

        echo "Vertical inversion at note $counter"
        simfind -s -b0 -a0 "$melody1 b2flp2.dms

        echo "Horizontal inversion at note $counter"
        simfind -s -b0 -a0 "$melody2" b2flp2.dms

        counter=`expr $counter + 1`
done
```

The variables 'melody1' and 'melody2' contain the fugue subject after vertical and horizontal inversion, respectively. The length of the fugue subject, calculated using the *density* tool, is assigned to the variable 'length', and the 'while' loop runs the *simfind* tool on the fugue subjects which have been inverted horizontally and vertically using each note in turn as the axis for inversion. The inverted fugue subjects have only been used in their original key. In order to use them in every key as well, a nested loop must be inserted into the above program to transpose them up a distance of one to thirteen semitones. Such an amended shellscript program could be as follows:

```
length=`density -s b2f1s.dms`

counter=1
while [ "$counter" -le "$length" ]
do
 distance=1
 while [ "$distance" -le 13 ]
 do
  melody1=`invert -n$counter b2f1s.dms | transpose -u$distance`
  melody2=`invert -rn$counter b2f1s.dms | transpose -u$distance`

  echo "Vertical inversion:$counter; transposition:$distance"
  simfind -s -b0 -a0 "$melody1" b2f1p2.dms

  echo "Horizontal inversion:$counter; transposition:$distance"
  simfind -s -b0 -a0 "$melody2" b2f1p2.dms

  distance=`expr $distance + 1`
 done
done
```

The inner loop uses the *simfind* tool on each transposed version of the inverted fugue subject before the outer loop generates a new inversion using a different axis note. To achieve the transposition of an inverted fugue subject, the data file 'b2f1s.dms' is first inverted using the *invert* tool, piped through the *transpose* tool, and then assigned to a variable. This program, even though it generates a great deal of output, only searches the second part of the first fugue for subject material. If all parts are to be searched, either the data-file names need to be changed and the program re-executed for each part, or the whole shellscript

178

program needs to be enclosed in yet another loop which will try each fugue part in turn.

The *motif* tool may be used in a similar fashion with transposed and inverted fugue subjects. The *motif* tool, however, is designed to locate elaborated (ie expanded) versions of a given melody, and the standard *motif* command may be enclosed in a set of nested shellscript loops similar to that of the previous example shellscript program.

The *fsets* tool, since its sets are deemed to be rest delimited, would seem to be of little use when attempting to search a fugue for fugue subject material. Most of the fugue subjects are continuous melodies with no rests, either in the middle or at the end of the subject. Although no tools in the Analysis Environment are available for finding rests in a string of DARMS data, the UNIX 'grep' tool may be used to display which fugue subjects contain rests by using the following command line:

```
grep "R" b2f*s.dms
```

The DARMS symbol for a rest is the letter 'R', and the grep command will display the names of data files which contain the letter 'R' (ie rests). The pattern 'b2f*s.dms' enables the 'grep' command to examine the data files 'b2f1s.dms', 'b2f2s.dms', 'b2f3s.dms' and so on. One way of 'fooling' the *fsets* tool is to edit the DARMS-encoded data file containing the fugue of interest and carry out two tasks. Firstly, remove all the rests from the fugue subject itself, and secondly, insert a rest at the end of the fugue subject. This procedure will turn the fugue subject into a set in its own right, and the *fsets* tool will pick up this set; and, the locations of other similar sets will be shown in relation to this set. Such a use of the *fsets* tool is likely to produce informative results only when a fugue subject is limited in the number of pitches it uses. A highly chromatic fugue subject will have so many pitches and produce such a large set that the majority of other

sets will almost certainly have some relationship with the fugue-subject set. To 'weed' out the fugue subjects containing too many different pitches, the *freq* tool can be used to count the total number of different pitches used. If the '-P' option is used, the *freq* tool will ignore the octave position of pitches in a similar fashion to that used by the *fsets* tool. When using the *freq* tool to count pitches, each pitch name is displayed on a separate line, together with the total number of occurrences of the pitch. If a particular pitch is not used, the line displaying its total number of occurrences is not displayed. Thus if an extract of DARMS-encoded data uses ten different pitches, when the *freq* tool is used with the '-P' option the total number of lines displayed will be ten. For example, the following command line displays the total occurrences of each pitch in the fugue subject of the first fugue,

```
$ freq -Ps b2fls.dms
Pitch usage:
A 2
C 1
D 2
E 4
F 7
G 5
$
```

and lines are only printed out for pitches which are used. Thus, using the UNIX 'wc' tool (which counts characters, words and lines) with the '-l' option to count lines, it is possible to display the number of lines produced by the above use of the *freq* tool and thus the number of different pitches employed in the fugue subject. The command line would have to be amended to:

```
$ echo "`freq -Ps b2fls.dms | wc -l` -1" | bc
6
$
```

The actual number of lines produced by the *freq* tool is greater than the number of pitches used because the first line contains a textual description of the

output. The total has been decremented by one, by piping the output through the UNIX 'bc' tool.

A shellscript program to apply the same *freq* and 'wc' tools on all fugue subjects would take the following format:

```
for subjectfile in `ls b2f*s.dms`
do
        total=`freq -Ps $subjectfile | wc -l`
        echo "$subjectfile: $total"
done
```

As before, the UNIX 'ls' command produces a list of data-file names which are then used in the 'for' loop. The *freq* and 'wc' tools are bolted together to produce the total number of different pitches, and this is assigned to the variable 'total'. Both the total and the data-file name are then printed out. The program prints out the total number of pitches in each data file (ie fugue subject) whether the fugue subject contains all twelve different pitches (which is too chromatic for use with the *fsets* tool) or just one pitch. Since seven pitches constitute the major or minor scale, this can be regarded as a suitable cutting off point, and the above shellscript loop may be adapted (by the insertion of an 'if' command) to display the data-file names and totals of fugue subjects with less than eight different pitches. Such an amended program would have the following syntax:

```
for subjectfile in `ls b2f*s.dms`
do
        total=`freq -Ps $subjectfile | wc -l`

        if [ "$total" -lt 8 ]
        then
                echo "$subjectfile: $total"
        fi
done
```

The fugues of the fugue subjects selected could then be subjected to the *fsets* tool (providing that the editing of the data files, previously described, has taken place).

Further analysis of the fugue subject may be undertaken in a comparative fashion. Information extracted from the fugue subject of the first fugue can be compared with similar information extracted from the other fugue subjects in an effort to discover a common stylistic structure. The *freq* tool, used earlier to determine the number of different pitches used, may also be used to procure a whole list of different statistical data such as the total number of bars and the total number of notes. Such information is likely to differ greatly from fugue subject to fugue subject because some subjects are lively and have many notes per bar and others are more lethargic and have fewer notes per bar. The length of a fugue subject might not be significant as an entity in its own right, but taken as a percentage of the fugue part might prove to be interesting. Such a feat can be achieved using the UNIX 'bc' tool which is a very sophisticated calculator. Sophistication often elbows out simplicity, and the 'bc' tool is no exception to this rule. It does, however, serve its purpose, and the following shellscript program, for example, will calculate the length of the first-fugue subject as a percentage of the second part:

```
subjectlength=`freq -ns b2f1s.dms`
partlength=`freq -ns b2f1p2.dms`
echo "scale=2; $subjectlength / $partlength * 100" | bc
```

The variables 'subjectlength' and 'partlength' are assigned the number of notes in the fugue subject and second part (achieved via the *freq* tool and the '-n' option) respectively. The echo command is used to pipe a scale (setting an accuracy of two decimal places for the calculation of the percentage) and the percentage calculation through the 'bc' tool, which sends the result to standard output, ie the screen. The same calculation may be performed, again using shellscript loops, on each fugue and fugue subject, but should really (for consistency) be performed on the part from which the fugue is extracted. Achieving the same calculations using the number of bars as opposed to the

number of notes is not a problem because the number of bars is consistent between parts whereas the number of notes is not.

Calculating the length of the fugue subject as a percentage of the whole fugue shows that generally, the subject is between four and six percent the size of the entire fugue.

This in turn brings us to the distribution of notes to parts. The total number of notes per part, and the total number of parts per fugue (which might prove to be an interesting method of comparison) may be achieved using the following shellscript program:

```
for fuguenumber in 1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24
do
        echo "Fugue: $fuguenumber"

        totalparts=`ls b2f${fuguenumber}p*.dms | wc -l`

        part=1
        while [ "$part" -le "$totalparts" ]
        do
                echo "Part $part: \c"
                freq -ns b2f${fuguenumber}p$part.dms

                part=`expr $part + 1`
        done
done
```

The *freq* tool may also be used to calculate the most frequent two and three-note phrases by using the '-2' and '-3' options respectively. This will provide interesting information if it is applied to the fugues as a complete corpus. To achieve this, all the fugues have to be transposed into the same key, concatenated together and then stored in the same data file. This is a straightforward task because the key of a fugue correlates with the fugue number. For instance, fugues one to six are in the keys C major, C minor, C# major, C# minor, D major and D minor respectively, and as such, a loop can be written which will progress through all the fugues, transposing them down by

183

the required amount into the key of C major. Such a loop would have the following format:

```
> fugues
distance=0
while [ "$distance" -le 11 ]
do
 fuguemaj=`expr $distance * 2 + 1`
 fuguemin=`expr $distance * 2 + 2`

 for part in 1 2 3 4
 do
 transpose -d$distance b2f${fuguemaj}p$part >>fugues 2>/dev/null
 done

 for part in 1 2 3 4
 do
 transpose -d$distance b2f${fuguemin}p$part >>fugues 2>/dev/null
 done

 distance=`expr $distance + 1`
done
```

The fugue in C# major (fugue three) has to be transposed down a semitone to put it into the key of C major, whereas the fugue in D major (fugue five) has to be transposed down two semitones. Thus, the number of semitones of movement correlates exactly with the fugue number, ie fugue 1 - down zero semitones, fugue 3 - down 1 semitone, fugue 5 - down 2 semitones, and fugue 7 - down 3 semitones etc. So the distance of transposition of a major fugue is (*fugue number* - 1) / 2. Although a minor fugue should not be transposed into a major key, it should (for this particular test) be transposed down by the requisite number of semitones to make its tonic the pitch 'C'. The distance of transposition of a minor fugue is (*fugue number* - 2) / 2, ie fugue six (in D minor) should be moved down (6-2)/2 semitones, which is two semitones. The first line of the program uses the redirection facility of UNIX to create an empty data file called 'fugues' by redirecting nothing into it. This will create the file if it does not exist, or overwrite its contents if it does exist. The new data-file 'fugues' will contain DARMS data for all the parts of all the fugues, transposed down so that the tonic of each is the same. At some point, it will prove to be

worthwhile if the major fugues are put into a separate data file from the minor fugues. This will offer a base of data which may be used to compare the style and structure of major fugues with minor fugues, The variable 'distance', used throughout the program, contains the number of semitones which the fugues are to be transposed down. The variable 'distance' is used in a 'while' loop which increments the variable from a starting distance of zero semitones (for the fugues in C major and C minor) to the final distance of eleven semitones (for the fugues in B major and B minor). The logic of the loop is based upon semitone distance, so the two variables 'fuguemaj' and 'fuguemin' are used to hold the current number of major and minor fugue respectively—calculated from the distance of transposition. Two 'for' loops are used within the 'while' loop to transpose the parts of the current major and minor fugue. The actual *transpose* command,

```
transpose -d$distance b2f${fuguemaj}p$part >>fugues 2>/dev/null
```

operates on parts one through to four (held in the variable 'part') and outputs into the data-file 'fugues' using the >> UNIX redirect facility. Not all fugues have four parts however, and an attempt to invoke the *transpose* tool on the data file 'b2f1p4' will produce an error message since the first fugue has only three parts. This is not a problem because the error message can be removed by again using the UNIX redirection facility. The syntax,

```
2>/dev/null
```

redirects the standard error stream (ie the error messages) into the file '/dev/null'. Everything, whether it is a printer, a disk drive, DARMS data, or a document, is deemed to be a file under UNIX. The file '/dev/null' is actually 'a hole in the back of the computer', and any output redirected to it 'simply disappears'.

Once the data file 'fugues' has been created, the *freq* tool may be applied to it in the following fashion,

```
freq -s23 fugues
```

to calculate the frequency of each two-note and three-note phrase used, in effect the most popular idiom.

With a simple amendment, the previous shellscript program may be used to put all the major fugues into one data file, and all the minor fugues into another data file ready for comparative analysis. The amendments involve the redirection of output from the *transpose* tool. In the first inner 'for' loop the output should be redirected using the following syntax:

```
transpose -d$distance b2f${fuguemaj}p$part >>majfugues 2>/dev/null
```

whilst in the second inner 'for' loop the output should be redirected using the following syntax:

```
transpose -d$distance b2f${fuguemaj}p$part >>minfugues 2>/dev/null
```

The *freq* tool can now be applied to both new data files and the statistics produced will be available for comparison. Without options, the *freq* tool will provide information on the frequency of accidentals, pitches, durations, and two and three-note phrases, and also determine the total number of notes and bars. If the number of notes and bars is divided by twelve, the average number of notes and bars for major and minor fugues will be obtained. The command lines for extracting statistics on major and minor fugues, via the *freq* tool, would be as follows:

```
freq -s majfugues
freq -s minfugues
```

where the '-s' option is used to suppress the output of the data from the DARMS-encoded data file. The average number of notes may be calculated using the following shellscript program:

```
for type in majfugues minfugues
do
        totalnotes=`freq -sn $type`
        totalbars=`freq -sb $type`

        echo "average $type notes=\c"
        echo "scale=2; $totalnotes / 12" | bc

        echo "average $type bars=\c"
        echo "scale=2; $totalbars / 12" | bc
done
```

Keys of fugues may be confirmed using the *key* tool. The *key* tool, however, has difficulty with highly chromatic melodies when determining what is and is not an essential accidental to the current key. In such an instance, the *key* tool describes the key as 'unknown'. This can, though, be used advantageously because the greater the number of 'unknowns' returned by the *key* tool, the more chromatic the melody is. In effect the 'unknown' quantity is a measure of the 'chromatic level' of the piece. The 'unknown' points in a fugue part may be calculated using the following shellscript:

```
key -ps b2f1p2.dms | awk '
BEGIN        {count=0; RS=" "}
/unknown/    {count++}
END          {print "chromatic level:", count}
'
```

The '-p' option invokes the progressive mode of the *key* tool where a new key is evaluated and displayed each time an accidental is used. The UNIX 'awk' tool (a pattern-matching language) assigns zero to a 'count' variable at the beginning of processing, increments the variable by one whenever the word 'unknown' is encountered, and prints out the total at the end of processing. To be of more value, the above routine should be tried for each part of a fugue, and an average calculated. This is necessary because those fugues with a

187

greater number of parts are likely to have a higher 'chromatic level' than those with fewer parts.

This chapter has not really produced real and visible results, merely ways of obtaining results. It was never the intention to provide a mound of statistics and graphs within the chapter. The main purpose of the chapter was to evaluate the usability of the <u>Analysis Environment</u> tools. Given a specific question, was it possible to integrate the tools with each other and the UNIX environment (using the shellscript programming language) and produce methods for solving the question? What this chapter has proved is that it is indeed possible to toss out an arbitrary question, pick up a handful of tools and bundle them together in a way which will procure new information from the score, and help the user to answer the question. The tools themselves do not provide the answer, they simply display information (ie the score) in a different format (eg statistic tables or shape graphs) and the user is required to make his or her own intuitive analytical steps in order to procure his or her answer. The intuitive steps might well involve connecting together other <u>Analysis Environment and</u> UNIX tools.

Conclusion

Like a box of Lego or a set of Meccano, there are thousands of ways in which to connect the <u>Analysis Environment</u> and UNIX tools together. Even when the relevant tools have been selected for a particular task, the very process of using the tools provokes an "I wonder what happens if I do this?" and a "which tools can I use to achieve it?" attitude, and the scholar thinks up new ideas and realises new goals as he or she progresses.

The more tools the better. Some particular types of analyses are difficult with the current set of <u>Analysis Environment</u> tools. The tools have to be combined in complicated ways in order to extract certain types of information. If there is a wide variety of tools, scholars are more likely to find a tool which is relevant to the current task.

There is no reason why new tools have to be geared toward analysis. Tools could be created for composition, such as the *shuffle* and *squash* tools. All that is required, is that the tools accepts DARMS data on the standard input, and send any altered DARMS data (if required) onto standard output. Any informative messages should be sent to the standard error stream. Using this method, the DARMS data can be piped into another tool, yet at the same time, informative messages can appear on the screen. Since DARMS data goes in to and out of the tools, there is no reason why (given that there is appropriate hardware available) a tool should not be able to take the DARMS as input and manipulate it in a compositional manner, eg play each pitch with a specific sound wave, or provide a bass line. Already, there are tools to invert and transpose melodies. Under MSDOS, there is even a tool to play melodies, albeit in a rather basic fashion.

If such an environment is to take off though, those writing and submitting tools for inclusion in the environment (a vetting procedure will weed out those

189

tools which are not unique or are non-standard) will have to program their tools to adhere to the standards. The environment needs to be shared. The environment should be freely available, with regular updates and new releases. The more people who use it, the more it will improve, grow and spread.

Appendix A (Analysis Environment Tools)

Command	Description
a11	'Analysis 11' command interpreter
anote	auxiliary-note remover
bend	melodic-phrase locator
cursatz	customised-ursatz finder
darmstrip	standard-DARMS stripper
density	melody-density calculator
endan	ending analyser
extract	DARMS music-code extractor
form	melody-form evaluator
freq	feature counter
fsets	Forte-set finder
help	manual printer
htod	Hewlett to DARMS converter
intfind	interval finder
invert	melody inverter
key	melody-key evaluator
motif	Reti-motif finder
play	DARMS music-code player
range	melody-range calculator
rcheck	repeated-note checker
rhythm	rhythm pattern-matcher
rnote	repeated-note remover
score	DARMS music-code notator
semio	paradigm finder and layout producer
shape	melody-shape printer
shuffle	melody shuffler
simfind	similarity pattern-matcher
squash	melody squasher
sync	syncopation measurer
transpose	melody transposer
ursatz	Schenker-Ursatz finder
vary	interval direction changer

NAME

a11 - 'Analysis 11' command interpreter

SYNOPSIS

a11 -f *program_name file_name*

DESCRIPTION

The **a11** command reads and executes 'Analysis 11' commands, applying them to a specified DARMS-encoded data file.

The **a11** command accepts the following options:

-f *program_name*   Reads commands from the file specified.

-s   Suppresses output of filtered data.

-v   Verbose mode. Prints commands during execution.

NAME

anote - auxiliary-note remover

SYNOPSIS

anote *options file_name*

DESCRIPTION

The **anote** command removes auxiliary notes from a DARMS-encoded data file. With no options specified, **anote** scans the data file once only, removing auxiliary notes.

An auxiliary note may be regarded as the middle note in a three-note melody where the first and last notes are the same pitch and the middle note is either a pitch a second higher or lower.

Auxiliary notes are selected pictorially, ie if the first and last notes of a three-note melody are on a line of the musical stave, a pitch higher is deemed to be anything in the space above, irrespective of accidental.

The **anote** command accepts the following options:

-r	Reiteratively scans the DARMS-encoded data file, removing all auxiliary notes.
-v	Verbose mode. Prints messages during execution.

SEE ALSO

rnote

NOTES

If the **anote** command is used on DARMS-encoded data files that include rhythmic data, all rhythmic data will be removed before the **anote** tool performs auxiliary note removal.

NAME

    bend - melodic-phrase locator

SYNOPSIS

    bend *options file_name*

DESCRIPTION

The **bend** command displays the beginning and ending positions of melodic phrases within a DARMS-encoded data file. Melodic phrases are deemed to be rest-delimited melodies.

With no options specified, **bend** displays the beginning and ending note-number for each melodic phrase.

The **bend** command accepts the following options:

-b	Displays the beginning positions only.
-e	Displays the ending positions only.
-m	Displays the beginning and ending positions as a multiple of the smallest rhythmic unit in the DARMS-encoded data file.
-n*number*	Displays the beginning and ending positions for melodic-phrase number 'number'.
-s	Suppresses output of filtered data.
-v	Verbose mode. Prints messages during execution.

NAME

cursatz - customised-Ursatz finder

SYNOPSIS

cursatz -vs -u "*DARMS ursatz*" *file_name*

DESCRIPTION

The **cursatz** command displays possible locations of user-created 'Ursaetze' within a DARMS-encoded data file. With no options specified, **cursatz** locates pitch sequences in the same octave position as the pitch sequence specified on the command line.

The customised 'ursatz' may comprise any set of pitches.

The **cursatz** command accepts the following options:

-o   Ignores octave position of pitches.

-s   Suppresses output of filtered data.

-v   Verbose mode. Prints messages during execution.

SEE ALSO

ursatz

NAME

>   **darmstrip** - standard-DARMS stripper

SYNOPSIS

>   **darmstrip** *options file-name*

DESCRIPTION

The **darmstrip** command strips a DARMS-encoded data file of all unnecessary information—ie data not required by the analysis environment tools. Data remaining after use of the **darmstrip** command includes pitch, rhythm, rest and barline data.

With no options specified, **darmstrip** hard-codes the sharps and flats in the key signature into the resultant data.

The **darmstrip** command accepts the following options:

-k	Ignores key signature. Does not hard-code key signature into resultant data.
-v	Verbose mode. Prints messages during execution.

SEE ALSO

htod

NAME

> **density** - melody-density calculator

SYNOPSIS

> **density** *options file_name*

DESCRIPTION

The **density** command counts the total number of notes in a DARMS-encoded data file.

The **density** command accepts the following options:

-b   Calculates the average number of notes per bar.

-s   Suppresses output of filtered data.

-v   Verbose mode. Prints messages during execution.

NAME

    endan - ending analyser

SYNOPSIS

    endan *options file_name*

DESCRIPTION

The **endan** command counts features and displays statistical data on the last five bars of a DARMS-encoded data file. With no options specified, **endan** displays all information possible on the ending of the DARMS-encoded data file.

The **endan** command accepts the following options:

-d	Displays the distance between the highest and lowest notes.
-m	Displays the melodic movement throughout the last five bars.
-N	Displays the total number of notes employed during the last five bars.
-n	Displays the total number of notes for each bar.
-R	Displays the total number of repeated notes employed during the last five bars.
-r	Displays the total number of repeated notes for each bar.
-s	Suppresses output of filtered data.
-v	Verbose mode. Prints messages during execution.

SEE ALSO

freq, extract

NAME

extract - DARMS music-code extractor

SYNOPSIS

extract *options file_name*

DESCRIPTION

The **extract** command extracts sequences of bars or sequences of notes from a DARMS-encoded data file. With no options specified, **extract** extracts the complete data file.

The **extract** command accepts the following options:

-b*numb,numb2*   Extracts bar number *numb*, or bars numbered from *numb* to *numb2*.

-n*numb,numb2*   Extracts note number *numb*, or notes numbered from *numb* to *numb2*.

-t*numb,numb2*   Extracts beat number *numb*, or beats numbered from *numb* to *numb2*. Beat numbers must be a multiple of the smallest rhythmic unit in the DARMS-encoded data file.

-v   Verbose mode. Prints messages during execution.

SEE ALSO

htod

NOTES

The b, n and t options cannot be used together.

The variables 'numb' and 'numb2' have an upper limit of 32767.

NAME

   form - melody-form evaluator

SYNOPSIS

   form *options file_name*

DESCRIPTION

The **form** command evaluates a formal structure for a DARMS-encoded data file, employing standard ABA notation. With no options specified, **form** takes accidental and octave data into account.

The **form** command accepts the following options:

-a	Ignores accidentals.
-k	Produces a key to aid identification of suggested patterns.
-o	Ignores octave position of notes.
-s	Suppresses output of filtered data.
-v	Verbose mode. Prints messages during execution.

NAME

>   freq - feature counter

SYNOPSIS

>   freq *options file_name*

DESCRIPTION

The **freq** command counts the occurrences of specified items in a DARMS-encoded data file. With no options specified, freq counts the occurrences of all items.

The **freq** command accepts the following options:

-2	Calculates the frequency of each two-note phrase used.
-3	Calculates the frequency of each three-note phrase used.
-a	Calculates the frequency of each 'black note' used.
-b	Calculates the total number of bars.
-n	Calculates the total number of notes.
-P	Calculates the frequency of each pitch used, disregarding octave position.
-p	Calculates the frequency of each pitch used.
-r	Calculates the frequency of each duration used.
-s	Suppresses output of filtered data.
-v	Verbose mode. Prints messages during execution.

SEE ALSO

endan

NAME

      fsets - Forte-set finder

SYNOPSIS

      fsets -vs *file_name*

DESCRIPTION

The **fsets** command displays information on the serial sets used within a DARMS-encoded data file. Rests are used as set delimiters.

The **fsets** command accepts the following options:

-s   Suppresses output of filtered data.

-v   Verbose mode. Print messages during execution.

NAME

      help - manual printer

SYNOPSIS

      help *command_name*

DESCRIPTION

The **help** command displays a page from the manual on a selected command. With no options specified, **help** displays a list of all commands available.

NOTES

Specifying a -? option with any command will display a list of valid options.

NAME

htod - Hewlett to DARMS converter

SYNOPSIS

htod *options file_name*

DESCRIPTION

The **htod** command converts a Hewlett-encoded data file into DARMS format. With no options specified, **htod** ignores rhythmic data.

The **htod** command accepts the following options:

-r   Includes rhythmic data during conversion process.

-v   Verbose mode. Prints messages during execution.

NOTES

Hewlett-encoded data files should only contain one musical part.

NAME

intfind - interval finder

SYNOPSIS

intfind *options* -i *interval file_name*

DESCRIPTION

The intfind command locates the occurrences of a user-selected interval within a DARMS input file. With no options specified, intfind displays all identical occurrences of the user-selected interval.

The intfind command accepts the following options:

-f	Displays only the first occurrence of the selected interval.
-N*number*	Displays all occurrences of the user-selected interval where exactly *number* notes separate the first and last notes of the interval.
-n*number*	Displays all occurrences of the user-selected interval where up to *number* notes separate the first and last notes of the interval.
-s	Suppresses output of filtered data.
-v	Verbose mode. Prints messages during execution.

NOTES

-i*interval*

The above method should be used when selecting the interval to be located, where *interval* may comprise the following:

1–15	interval size
a	augmented
d	diminished
m	minor

Thus, -i7d selects a diminished seventh for the search.

NAME

invert - melody inverter

SYNOPSIS

invert *options file_name*

DESCRIPTION

The **invert** command inverts a DARMS-encoded melody. With no options specified, **invert** uses the middle note of the melody as the axis for inversion, and turns the melody upside down. If a melody comprises an even number of notes, the melody is inverted using the first of the middle two notes as the axis for inversion.

The **invert** command accepts the following options:

-n*number*	Changes the *number* of the note used as the axis for inversion.
-r	Inverts the melody horizontally, ie turns it back-to-front.
-v	Verbose mode. Prints messages during execution.

SEE ALSO

transpose

NAME

key - melody-key evaluator

SYNOPSIS

key *options file_name*

DESCRIPTION

The **key** command will determine the overall key of a DARMS data file.

The **key** command accepts the following options:

-b	Evaluates and prints the key for each bar.
-p	Progressive mode. Evaluates and prints the new key every time an accidental is used or cancelled.
-s	Suppresses output of filtered data.
-v	Verbose mode. Prints messages during execution.

NOTES

The p and b options cannot be used together.

NAME

motif - Reti-motif finder

SYNOPSIS

motif *options* -m "*DARMS motif*" *file_name*

DESCRIPTION

The **motif** command displays a DARMS-encoded data file in terms of the "DARMS motif" entered on the command line. With no options specified, the **motif** command displays the entire DARMS-encoded data file and shows, on a line below the original melody, how the melody might have evolved from the "DARMS motif".

The **motif** command accepts the following options:

-n*number*   Limits the amount of notes used in evolved material to *number.* eg -n10 states that the motif must not be shown to evolve to more than 10 notes.

-s   Suppresses output of filtered data.

-v   Verbose mode. Prints messages during execution.

NAME

      play - DARMS music-code player

SYNOPSIS

      play *options file_name*

DESCRIPTION

The **play** command produces an audible rendition of a DARMS-encoded data file.

The **play** command accepts the following options:

-n	Creates a noise by playing each note of the DARMS data file with a duration of zero seconds.
-r	Uses rhythmic data.
-s	Suppresses output of filtered data.
-v	Verbose mode. Prints messages during execution.

SEE ALSO

score, shuffle, squash, vary

NOTES

The **play** command is only available within the MSDOS environment.

NAME

    range - melody-range calculator

SYNOPSIS

    range *options file_name*

DESCRIPTION

The **range** command calculates the distance between the highest and lowest notes of a DARMS-encoded data file. With no options specified, **range** returns a 'lines and spaces' distance.

The **range** command accepts the following options:

-c   Calculates semitone distance between highest and lowest notes.

-s   Suppresses output of filtered data.

-v   Verbose mode. Prints messages during execution.

NAME

   rcheck - repeated-note checker

SYNOPSIS

   rcheck *options file_name*

DESCRIPTION

The **rcheck** command examines the repeated notes of a DARMS-encoded data file. With no options specified, **rcheck** calculates the percentage of repeated pitches and durations within a DARMS-encoded data file.

The **rcheck** command accepts the following options:

-c	Counts the repetitions of each duration used.
-C	Counts the repetitions of each pitch used.
-g	Displays a graph of repeated duration activity.
-G	Displays a graph of repeated pitch activity
-s	Suppresses output of filtered data.
-v	Verbose mode. Prints messages during execution.

SEE ALSO

rnote

NOTES

The DARMS input file must contain rhythm data.

NAME

> rhythm - rhythm pattern-matcher

SYNOPSIS

> rhythm *options* -r "rhythm pattern" *file_name*

DESCRIPTION

The rhythm command searches a DARMS-encoded data file for the occurrences of a selected rhythm pattern. With no options specified, rhythm displays identical matches.

The rhythm command accepts the following options:

-c	Locates compressed version of the rhythm pattern.
-s	Suppresses output of filtered data.
-t	Locates stretched versions of the rhythm pattern.
-v	Verbose mode. Prints messages during execution.

NAME

rnote - repeated-note remover

SYNOPSIS

rnote *options file_name*

DESCRIPTION

The **rnote** command removes repeated notes from a DARMS-encoded data file. With no options specified, **rnote** scans the data file once only, removing repeated notes. New repeated notes might be created using **rnote** without options.

The **rnote** command accepts the following options:

-r   Reiteratively scans the DARMS data file, removing
                      all repeated notes.

-v   Verbose mode. Prints messages during execution.

SEE ALSO

anote, rcheck

NOTES

If the **rnote** command is used on DARMS-encoded data files that include rhythmic data, all rhythmic data will be removed before the **rnote** tool performs repeated note removal.

NAME

score - DARMS music-code notator

SYNOPSIS

score -v *file_name*

DESCRIPTION

Using the standard ASCII character set, the **score** command displays a DARMS-encoded data file in a pseudo-musical notation.

The **score** command accepts the following option:

-r   Uses DARMS rhythm letters for note-heads.

-v   Verbose mode. Prints bar numbers.

SEE ALSO

play, shape

NOTES

The **score** command does not print rhythmical data.

NAME

semio - paradigm finder and layout producer

SYNOPSIS

semio *options file_name*

DESCRIPTION

The **semio** command displays a DARMS-encoded data file in a chronological set of columns, each containing a list of similar melodies. With no options specified, **semio** accounts for the entire DARMS-encoded data file when creating its columns, and searches for melodies one note in length.

The **semio** command accepts the following options:

-n*number*	Limits the size of melodies to *number* notes in each column.
-s	Suppresses output of filtered data.
-v	Verbose mode. Prints messages during execution.

NAME

   shape - melody-shape printer

SYNOPSIS

   shape *options file_name*

DESCRIPTION

The **shape** command displays the overall shape of a DARMS-encoded melody using standard ASCII characters.

The **shape** command accepts the following options:

-m	Smooths the overall shape. eg a slight fall in an overall rising melody would smoothed out to produce just a rising melody.
-r	Reduces the overall shape to 'up's and 'down's. eg five rising intervals would be reduced to a single rising interval.
-s	Suppresses output of filtered data.
-v	Verbose mode. Prints messages during execution.

SEE ALSO

score

NAME

  shuffle - melody shuffler

SYNOPSIS

  shuffle *options file_name*

DESCRIPTION

The **shuffle** command changes the order of bars or note-units within a DARMS-encoded melody. With no options specified, **shuffle** puts the bars of a DARMS-encoded data file into an arbitrary order.

The **shuffle** command accepts the following options:

-n *number*	Shuffles the order of *number*-note groups. For example, -n3 will shuffle the order of 3-note groups.
-c	Ensures that the end of a shuffled note group connects with adjacent note-groups, ie the last pitch of a note group will be identical to, or move by up to a major third to the first pitch of the next note group.
-v	Verbose mode. Prints messages during execution.

SEE ALSO

play, squash, vary

NAME

 simfind - similarity pattern-matcher

SYNOPSIS

 simfind *options* -b *beta* -a *alpha* -m *"melody" file_name*

DESCRIPTION

The **simfind** command locates occurrences of a specified melody within a DARMS-encoded data file. With no options specified, **simfind** displays all similar located melodies in order of similarity.

The **simfind** command accepts the following options:

-c   Displays the located melodies in chronological order.

-f   Displays the set of non-overlapping located melodies whose total similarity is greatest.

-s   Suppresses output of filtered data.

-v   Verbose mode. Prints messages during execution.

NOTES

The *beta* value is used to specify the importance of the difference between real and tempered frequencies. The higher the value, the greater the importance. *Beta* may take any value greater than zero and less than or equal to one hundred.

The *alpha* value is used to specify the importance of taking dissonance into consideration. The higher the value, the greater the importance. *Alpha* may take any value greater than zero and less than or equal to one.

If either *alpha* or *beta* are set to zero, their use within the similarity formula is ignored.

NAME

    squash - melody squasher

SYNOPSIS

    squash *options file_name*

DESCRIPTION

The **squash** command augments or diminishes the size of intervals within a DARMS-encoded data file. With no options specified, **squash** diminishes all intervals by a semitone.

The **squash** command accepts the following options:

-a	Augments the size of intervals.
-n *number*	Specifies the *number* of semitones by which the intervals should be augmented or diminished.
-v	Verbose mode. Prints messages during execution.

SEE ALSO

play, shuffle, vary

NAME

    sync - syncopation measurer

SYNOPSIS

    sync *options file_name*

DESCRIPTION

The **sync** command calculates the percentage of notes which sound across strong beats, but are not 'struck' on strong beats. With no options specified, sync calculates the percentage of notes which sound across the first beat of a bar.

The **sync** command accepts the following options:

-b *beats*	Specifies which *beats* to check for notes.
-g	Draws a graph showing the percentage against time.
-p	Calculates a percentage for each bar.
-s	Suppresses output of filtered data.
-v	Verbose mode. Prints messages during execution.

NOTES

Beats may be specified either as comma separated numbers, eg 1,2,4 (meaning beats one, two and four), or as hyphen separated numbers, eg 1-3 (meaning beats one to three), or as a mixture, eg 1-3,5 (meaning beats one to three, and five).

The DARMS input file must contain rhythm data.

NAME

transpose - melody transposer

SYNOPSIS

transpose *options file_name*

DESCRIPTION

The **transpose** command transposes a DARMS-encoded melody up or down by a specified number of semitones. With no options specified, **transpose** leaves the melody in its original key.

The **transpose** command accepts the following options:

-d*number*   Transposes the melody down by *number* semitones.

-u*number*   Transposes the melody up by *number* semitones.

-v   Verbose mode. Prints messages during execution.

NAME

> **ursatz** - Schenker-Ursatz finder

SYNOPSIS

> **ursatz** *options* -k *key_name file_name*

DESCRIPTION

The **ursatz** command displays possible locations of 8-1, 5-1, 3-1 standard and prolonged Ursaetze within a DARMS-encoded data file. With no options specified, **ursatz** displays all locations of all Ursatz types.

The **ursatz** command accepts the following options:

-3	Displays standard 3-1 Ursaetze.
-5	Displays standard 5-1 Ursaetze.
-8	Displays standard 8-1 Ursaetze.
-p	Displays prolonged Ursaetze.
-s	Suppresses output of filtered data.
-v	Verbose mode. Prints messages during execution.

SEE ALSO

cursatz

NOTES

key_name may comprise the following:

A-G	letter-name of key
-	flat
#	sharp
m	minor (default is major)

Thus, -kA-m represents A flat minor.

NAME

> vary - interval direction changer

SYNOPSIS

> vary *options file_name*

DESCRIPTION

The **vary** command changes the direction of intervals within a DARMS-encoded data file. With no options specified, **vary** inverts all intervals.

The **vary** command accepts the following options:

-a   Only inverts ascending intervals.

-d   Only inverts descending intervals.

-v   Verbose mode. Prints messages during execution.

SEE ALSO

play, shuffle, squash

Appendix B (Analysis 11)

In much the same way that MUSIC 11 (a sound-synthesis programming language) can be manipulated to produce any sound a user imagines, a programming language which can be manipulated to produce any 'analysis' a user imagines would seem to be a realistic proposition. If inexplicable sounds can be apparently replicated using mathematical rules, inexplicable analytical questions can be answered using mathematical rules. This appendix outlines some basic research into the conception of a self-contained programming language for music analysis, the commands of which may be used in the Analysis Environment via the *a11* tool.

Since the inspiration for an analytical programming language arose from using MUSIC 11, the analytical programming language described in this appendix has been called ANALYSIS 11, and thus the tool in the Analysis Environment has been called *a11*.

The operation of MUSIC 11 requires two files, known as the 'score' and the 'orchestra'. In simple terms, the 'score' contains frequencies and durations, whilst the 'orchestra' is a program defining exactly how those frequencies and durations should be employed. ANALYSIS 11 also requires two files similar to MUSIC 11's 'score' and 'orchestra'. The 'score' for ANALYSIS 11 is, as one might well expect, a file containing an alphanumeric encoding of the music score to be subjected to analysis. The ANALYSIS 11 'orchestra' is a straightforward program, written by the user, employing ANALYSIS 11 commands to analyse and extract information from the 'score'.

MUSIC 11 does not operate in 'real time', ie both 'score' and 'orchestra' must be processed before any sound can be heard. This processing time may take anything between a few seconds and several hours to complete, dependent upon the complexity and length of both 'score' and 'orchestra'.

ANALYSIS 11, however, is an interpreted language where each line of a program or 'orchestra' is executed as soon as the computer reads it. Thus, even a complex and lengthy program will display results instantly on screen, allowing the possible abortion of a program run if the results do not appear as intended. A MUSIC 11 run may take several hours, and then produce garbage as a result of some simple error within the 'score'. Therefore, interpretation, rather than preprocessing or compilation, allows quick, easy trapping and correction of errors.

At first glance, SPITBOL — a programming language used in linguistics for its string handling capabilities — seemed an ideal medium in which to write the ANALYSIS 11 interpreter. Since basic pattern searching and pattern matching commands are already contained within SPITBOL, it was a straightforward task to use them as a basis for the interpreter. The ANALYSIS 11 interpreter has since been rewritten in C, which has increased the speed of execution of ANALYSIS 11 programs, and made the interpreter and programs compatible with the tools of the Analysis Environment. ANALYSIS 11 commands have been limited to searching and matching, definition of patterns, selection of output, and method of score translation.

The interpreter reads one line of an ANALYSIS 11 program at a time. If the line contains a command, the command is executed. Lines which contain comments or remarks are simply ignored.

Any syntax errors discovered by the interpreter will be displayed on the screen, together with a description of the error and its position within the program. If the interpreter can carry on, it will attempt to do so, otherwise the interpreter will stop and allow the user to edit the spurious line within the ANALYSIS 11 program.

The running of the ANALYSIS 11 interpreter may be aborted at any stage by pressing the 'interrupt' key on the terminal's keyboard, and leaves the user in the local operating system to allow editing of the ANALYSIS 11 program.

Any program line may be given a label, and, as in other programming languages, sections of a program can be repeated or skipped by instructing the computer to return to, or go to a specific label within the program.

Before a musical score can be analysed under ANALYSIS 11, it must be translated into an alphanumeric code. (At present, ANALYSIS 11 will only operate on the subset of the DARMS encoding language outlined in chapter three.) The encoded musical score should be stored in the 'score' file of ANALYSIS 11, ready for analysis.

Running ANALYSIS 11 will reveal any errors in the 'score' file and also any nonsensical musical information such as bars which contain an incorrect number of beats. Although this restriction to 'orthodox' notation limits ANALYSIS 11 analysis to compositions of a certain type, it does however mean that ANALYSIS 11 can be used as a simple alphanumeric code checker.

The 'orchestra', or program file of ANALYSIS 11, contains a series of instructions to be obeyed by the computer. Each line of a program must follow a certain syntax. Any line may be given a label, but the label must start in the first column of the line to which it refers. Line labels may be composed of letters and numbers in a mixture of upper and lower case.

The instruction, or command, may occur at any point after the label. If a line has no label, however, the command on that line must not begin in column one, otherwise it will be regarded as a label.

Comments may be inserted anywhere in a program, either on a separate line, or after a command. Comments may be distinguished from labels and commands by preceding the comments with a semicolon.

The available ANALYSIS 11 commands fall into four categories: flow of control, definition, analysis and output.

The END Command

The **END** command must be the last line of any program, and informs
the interpreter when the end of the program has been reached.

syntax:

```
program line
program line
program line
END
```

The GOTO Command

The **GOTO** command sends the interpreter forwards or backwards to the label specified. The **GOTO** command is conditional, ie it will only operate if the last **LOCATE** command succeeded in finding a pattern.

syntax:

```
label     program line
          program line
          program line
          GOTO label
```

The START Command

The **START** command instructs the interpreter to start operating on the data held in the 'score' file at a specified bar, beat and note. The first parameter of the **START** command represents the bar number, whilst the second represents the crotchet beat number and the note within the beat, ie a decimal number where the 'whole' part signifies the beat and the 'decimal' part signifies the nth note of that beat.

syntax:

```
START bar number, beat number.note number
```

example:

```
START 2,3.1
```

explanation:

The above example starts processing the data from the first note of beat three, in the second bar.

The STOP Command

When the interpreter reaches the position in the 'score' file which was specified in the most recent **STOP** command, it will abort the current **LOCATE** command and proceed to the next program line.

The **STOP** command uses the same parameters as the **START** command.

syntax:

```
STOP bar number, beat number.note number
```

example:

```
STOP 34,4.2
```

explanation:

The above example informs the **LOCATE** command to stop processing the data at the second note of beat four, in the thirty-fourth bar.

The DEFINE Command

A pattern which is to be used frequently throughout a program may be given a label using the **DEFINE** command. Thus, whenever the pattern is required, its label may be used instead. For example, if a pattern is given the label 'notes', it may appear in a **LOCATE** command either in its literal form, or as "**LOCATE** *notes*," etc. Pattern labels may only contain lower case alphabetical characters.

syntax:

```
DEFINE pattern label, music pattern
```

example:

```
DEFINE phrase, 22# 20 25- 24
LOCATE phrase, INTERVAL
LOCATE phrase, OCTAVE
```

explanation:

The above example gives the label 'phrase' to the DARMS string '22# 20 25- 24'[117]. The label, and thus the actual DARMS string, is used in the two **LOCATE** commands (see over for a description of the **LOCATE** command) following the **DEFINE** command.

---

[117] See chapter three, starting on page 72, for a description of the DARMS subset used by the tools (including a11) of the <u>Analysis Environment</u>.

The LOCATE Command

The **LOCATE** command searches the 'score' data in an effort to find a specific pattern.

The **LOCATE** command consists of two parameters which define the pattern to be searched for and the type of search.

The first parameter, defining the pattern to be searched for, is mandatory. This pattern may occur in a literal form, ie individual notes entered at the first parameter position; or, it may be replaced by a label (but only if the label has previously had a pattern assigned to it with the **DEFINE** command).

Parameter two determines the search type. The default, IDENTICAL, allows the **LOCATE** command to succeed only if a pattern is found which is identical to the pattern being searched for. If an OCTAVE search is used, the **LOCATE** command will allow pattern matching between the search pattern and the same pattern appearing in a different octave. INTERVAL allows pattern matching between the search pattern and a pattern which uses the same intervals between notes, ie A B C# would match with C D E and Eb F G etc.

Every parameter must be separated by a comma. If a parameter is left out, its default value is assumed. However, even if a parameter is left out, its associated comma must still be present, ie "**LOCATE** *pattern*," is equivalent to "**LOCATE** *pattern*,IDENTICAL".

syntax:

```
LOCATE search pattern,search type
```

example:

```
LOCATE 25- 24 26 25, INTERVAL
```
             | |

(literal/label)   (IDENTICAL/OCTAVE/INTERVAL)

explanation:

The above example searches the 'score' data file for a pattern with the same intervals between notes as the DARMS string '25- 24 26 25'.

The SELECT Command

The **SELECT** command allows all key signatures and accidentals to be ignored during a program run. **SELECT** may be used any number of times, and at any point within a program. **SELECT** defaults to one, and may be given one of the following values:

	accidentals	key signature
1	on	on
2	on	off
3	off	off
4	off	on

syntax:

```
SELECT number
```

example

```
SELECT 3
```

explanation:

The above example forces any further pattern-matching commands to ignore both accidentals and key signatures within the 'score' data.

The TRANSFORM Command

The **TRANSFORM** command allows variation between the pattern found and the pattern being searched for. The pattern found may contain a different number of notes, as well as pitch variations, between itself and the pattern being searched for.

TRANSFORM is used to define the variation allowed between both pattern found and pattern searched for. This allowable variation must be given a lower case label. To employ the allowable variation within a **LOCATE** command, the label should be added to the pattern (or pattern label) contained within the **LOCATE** command.

syntax:

```
TRANSFORM label, variation_definition
LOCATE pattern+label,options
```

A **TRANSFORM** command must always appear before the **LOCATE** command in which it is to be employed.

The number of 'real' notes in a **TRANSFORM** command must equal the number of notes in a **LOCATE** command.

The remaining part of the **TRANSFORM** command contains the numbered notes of a search pattern together with their allowable variation. For example, the first notes of a search pattern may be allowed to vary by up to three semitones higher when occurring in a found pattern, in which case it is represented as:

, 1>3

, 1	note number
>	variation operator
3	variation limit

The variation operators are as follows:

>*n*   a note may vary by up to *n* semitones higher than what it should be.

<*n*   a note may vary by up to *n* semitones lower than what it should be.

+*n*   a note may only vary by *n* semitones higher.

-*n*   a note may only vary by *n* semitones lower.

Variation operators may be combined:

```
,2+3<10
```

The above example indicates that the second note of a found pattern may either occur exactly three semitones higher, or up to ten semitones lower than the second note of the search pattern.

If a note number is not contained within a variation definition, the definition is assumed to relate to all the notes of the pattern being searched for, and thus is not classified as a 'real' note within a variation definition.

Note-numbered variation definitions must occur in the order in which they are numbered.

```
1+2-4,3<1,2,4+1        is incorrect

1+2-4,2,3<1,4+1        is correct
```

The insertion of **n* indicates that the found pattern may contain up to *n* more notes than the search pattern. For example, searching for a pattern "A B", with a **TRANSFORM** definition of "1,*,2", may find a pattern "A C# B"; likewise, "1,*4,2" may find a pattern "A C# Eb D F B" or simply "A C# B". Extra notes in

the found pattern may occur only at the points marked in the TRANSFORM
definition with an asterisk.

example:

```
TRANSFORM label,+2
```

| +2 | Any note within a found pattern may occur exactly a tone above a note in an equivalent position within the search pattern. |

example:

```
TRANSFORM vari,1<2>4,2+1,3,4,*2,(5)
```

,1<2>4	The first note of the found pattern may be up to two semitones lower or up to semitones higher than the first note of the search pattern.
,2+1	The second note of the found pattern may only be the same or a semitone higher than the second note of the search pattern.
,3,4	The third and fourth notes of the found pattern must be identical in pitch to the third and fourth notes of the search pattern.
,*2	Up to two extra notes may occur between the fourth and fifth notes of the found pattern, ie the search pattern contains only five notes, but the found pattern may contain between five and seven notes, with the extra notes between note four and note five relative to the search pattern.
,(5)	The fifth note of the search pattern need not occur in the found pattern.

The MAP Command

The **MAP** command may occur at any point within a program. **MAP** is used to switch on graphical output to show the location of successful pattern matches.

MAP must be followed by the word ON or OFF and is initially set to OFF.

example:

```
MAP  ON
```

example output:

```
        Start                              Finish
Occurrences+     ++    +   +        +++    +
```

The PRINT Command

The **PRINT** command will output whatever string of characters is held within its set of speech marks. Since a set of speech marks defines the string to be output, it is not possible to have speech marks as part of the output string.

String lengths must be short enough to allow both the **PRINT** command, and its associated string, to fit on one line of the screen.

example:

```
PRINT "Normal pattern match"
```

explanation:

The above example prints the text 'Normal pattern match' onto the screen whenever the interpreter reaches the **PRINT** command.

The **LOCATE, DEFINE, TRANSFORM** and **SELECT** commands send operating messages to the screen whenever they are used. Textual output, other than that specified within a **PRINT** command, may be switched on or off with the **TEXT** command.

TEXT must be followed by the word ON or OFF and is initially set to ON.

example:

```
TEXT OFF
```

Sample Program

```
        DEFINE pattern, 26 21 20 21 26 21        ; (1)
        TRANSFORM vari, 1, 2<2, 3>2, 4<2, 5, 6   ; (2)
        SELECT 1                                 ; (3)
        MAP ON                                   ; (4)
        TEXT ON                                  ; (5)
        START 2, 1.1, 1                          ; (6)
        STOP 12, 1.1, 1                          ; (7)


        PRINT "Sample run..."                    ; (8)
        PRINT "Normal:"                          ; (8)
        LOCATE pattern, IDENTICAL                ; (9)


        PRINT "With variation"                   ; (8)
loop    LOCATE pattern+vari, INTERVAL            ; (10)
        GOTO loop                                ; (11)


        END                                      ; (12)
```

Program Commentary

; (1)   Stores the pattern "26 21 20 21 26 21" under the label "pattern".

; (2)   Stores a transform definition under the label "vari", which will allow the second, third and fourth notes of a found pattern to differ by up to a tone lower, higher and lower respectively, from the corresponding notes of a search pattern.

; (3)   Allows key signatures and accidentals to act on the data (default).

; (4)   Switches on the occurrence location map.

; (5)   Selects textual output (default).

; (6)   Starts analysis from the first note of bar two.

; (7)   Ends analysis at the first note of bar twelve.

; (8)   Prints out a text string.

; (9)   Searches for the first melodic repetition of the pattern labelled "pattern".

; (10)   Searches for all occurrences (in conjunction with the following **GOTO** command) of the pattern labelled "pattern" which vary in accordance with the transform definition labelled "vari".

; (11)   Goes back to the statement labelled "loop". Does not go back if the most recent **LOCATE** command has not found a pattern match.

; (12)   Mandatory **END** command.

Shellscript programming

The shell programming language is interpreted, ie each line in a program is analysed by the computer and then executed. Most other programming languages like C, Pascal, and FORTRAN are compiled, ie all the lines in a program are turned into a machine-executable form before execution.

Command files

Shellscript programs may either be typed directly at the computer's prompt, or put into a file and executed at a later date.

Directly:

```
$ who | wc -l
4
$
```

In a file:

users

```
echo There are...
who | wc -l
echo users on the system.
```

```
$ users
There are...
4
users on the system.
$
```

Any operating system commands may be put inside a file.

Comments

Remarks or comments may be inserted into a program to make it more readable and maintainable.

Any text after the special character #, and up to the end of the line, will be treated as a comment and simply ignored.

users

```
# A program to display the number of people
# on the system
echo There are...
who | wc -l                    # who piped through wc
echo users on the system.
```

Variables

Like the majority of programming languages, values may be stored in variables.

Variable names:

o   Must start with a letter or underscore (_).

o   May consist of letters, numbers, and underscores.

Values may be assigned to variables using an equals sign.

```
$ user_count=1
$ user_name=Clive
$
```

There should be no spaces round the equals sign.

Data Types

The shell has no concept of data types. All values assigned to variables are treated as strings of characters.

```
$ sum=2+4
$
```

The characters 2, +, and 4, are stored in the variable sum.

Displaying the Contents of Variables

When a variable name is preceded with the special character $, the shell substitutes the variable name for its contents.

```
$ user_count=1
$ user_name=Clive
$ echo $user_count
1
$ echo $user_name
Clive
$
```

```
$ echo my name is $user_name
my name is Clive
$
```

Unassigned variables

Variables which have not been assigned a value, contain the null value.

```
$ echo $nothing

$
```

To assign a null to a variable, either specify nothing or two speech marks on the right-hand side of the equals sign.

```
$ nothing=
$
```

or

```
$ nothing=""
$
```

Variable Substitution

When concatenating text to the contents of a variable, curly brackets
should be used to avoid confusion.

```
$ name=Cliv
$ correct_name=$namee
$ echo $correct_name
$
```

Using curly brackets:

```
$ name=Cliv
$ correct_name=${name}e
$ echo $correct_name
Clive
$
```

The curly brackets should enclose the entire variable name, but not the
leading dollar sign.

The quote characters

The shell recognises four different types of quote characters.

'	Apostrophe.
"	Speech mark.
\	Backslash.
`	Grave.

Apostrophe

Enclosing text inside two apostrophe characters forces the shell to ignore all special characters.

```
$ name=Clive
$ echo '$name *'
$name *
$
```

Speech mark

Enclosing text inside two speech marks forces the shell to ignore all special characters except dollar signs, graves, and backslashes.

```
$ name=Clive
$ echo "$name *"
Clive *
$
```

Backslash

The backslash character is equivalent to placing apostrophes around a single character.

```
$ name=Clive
$ echo "\$name *"
$name *
$
```

Grave

The shell executes text enclosed inside two grave characters, and replaces the text and the grave characters with the result.

```
$ echo the date is date
the date is date
$
```

```
$ echo the date is `date`
the date is Mon Jan  1 17:19:52 EDT 1990
$
```

Shell arithmetic

Since the shell has no concept of data types, it has no concept of

arithmetic.

```
$ a=1
$ b=2
$ c=$a+$b
$ echo $c
1+2
$
```

The command expr will evaluate an expression given to it on the

command line.

Each operator and operand passed to expr must be separated by

spaces.

```
$ expr 1+2
1+2
$
```

```
$ expr 1 + 2
3
$
```

```
$ a=1
$ b=2
$ c=`expr $a + $b`
$ echo $c
3
$
```

Doing arithmetic with decimals

The expr command only evaluates integer expressions. The command bc must be used to perform floating point calculations.

```
$ expr 1.2 "*" 1.4
expr: non-numeric argument
$
```

```
$ echo 1.2 "*" 1.4 | bc
1.6
$
```

To increase the number of decimal places given by bc, set the scale variable to the required number of decimal places.

```
$ echo "scale=2; 1.2 * 1.4" | bc
1.68
$
```

```
$ c=22
$ c=`echo "scale=10; $c / 7" | bc`
3.1428571428
$
```

Special variables

Positional parameters

Arguments can be passed from the shell command line to a shell program.

Whenever a shell program is executed, the shell automatically stores the first argument in the special shell variable 1, the second argument in the variable 2, and so on.

words

```
echo you typed $1 $2 $3
```

```
$ words the rain in Spain
you typed the rain in
$
```

The # variable

The # variable contains the number of arguments typed on the command line.

count

```
echo you typed $# words
```

```
$ count the rain in Spain
you typed 4 words
$
```

The * variable

The * variable contains all of the arguments typed on the command line.

words2

```
echo you typed $*
```

```
$ words2 the rain in Spain
you typed the rain in Spain
$
```

The shift command

The positional parameter variables only refer to nine command line arguments. If there is a tenth argument on the command line, the shift command must be used to access it.

The shift command will assign the contents of variable 2 to variable 1, variable 3 to variable 2, 4 to 3, and so on. The value of variable 1 will be lost.

The # variable will automatically be decreased by one.

shifty

```
echo you originally typed $1 $2 $3
shift
echo now it is $1 $2 $3
```

```
$ shifty the rain in Spain
you originally type the rain in
now it is rain in Spain
$
```

(The 0 variable contains the command itself.)

Making decisions

The ? variable

The ? variable contains the exit status of the last command executed. If the exit status is zero, the last command succeeded. If the exit status is nonzero, the command failed.

```
$ cd
$ echo $?
0
$ cd xxx
xxx: does not exist
$ echo $?
1
$
```

The if command

Syntax:

```
if command1
then
        command2
        command3
        . . .
fi
```

The if command executes *command1*. If the exit status of *command1* is zero, the commands between the then and the fi are executed. If the exit status of *command1* is nonzero, the commands between the then and the fi are skipped.

user

```
if who | grep "^$1"
then
        echo "$1 is logged on"
fi
```

```
$ who
clive           tty8a              Jan  1 07:14
$ user clive
clive           tty8a              Jan  1 07:15
clive is logged on
$ user elaine
$
```

(grep "^$1" selects the lines which contain the variable $1 at the beginning.)

The test command

Syntax:

```
test expression1
```

The test command evaluates *expression1*. If the result is true, test returns an exit status of zero. If the result is false, test returns a nonzero exit status.

String testing

The following operators are available:

operator	true if...
string1 = string2	string1 is identical to string2
string1 != string2	string1 is not identical to string2
string1	string1 is not null
-z string1	string1 is null

nametest

```
name=$1
if test $name = Clive
then
      echo "You typed my name."
fi
```

```
$ nametest Elaine
$ nametest Clive
You typed my name.
$
```

257

Integer testing

The following operators are available:

operator	true if...
integer1 -eq integer2	integer1 is equal to integer2
integer1 -ge integer2	integer1 is greater than or equal to integer2
integer1 -gt integer2	integer1 is greater than integer2
integer1 -le integer2	integer1 is less than or equal to integer2
integer1 -lt integer2	integer1 is less than integer2
integer1 -ne integer2	integer1 is not equal to integer2

File testing

The following operators are available:

operator	true if...
-d file	file is a directory
-f file	file is an ordinary file
-r file	file is readable
-s file	file has something in it
-w file	file is writable
-x file	file executable

The command...

```
$ test -d /usr/clive
$
```

returns true if /usr/clive is a directory.

Operators

The negation operator

An explanation mark can be placed in front of a test expression to negate the expression.

The command...

```
$ test ! -d /usr/clive
$
```

returns true if /usr/clive is not a directory.

The and operator

The -a operator can be used to join two expressions together and will return true if both the joined expressions are true.

The command...

```
$ test "$count" -gt 0 -a "$count" -lt 10
$
```

returns true if the variable count is greater than 0 and less than 10.

The or operator

The -o operator may also be used to join two expressions, but will return true if either the first expressions is true or the second expression is true.

The command...

```
$ test "$count" -lt 0 -o "$count" -gt 10
$
```

returns true if the variable count is either less than 0 or greater than 10.

An alternative to test

The test command...

```
test expression
```

may also be expressed as...

```
[ expression ]
```

Spaces must appear after the [and before the].

The program...

```
if test ! "$name" = "Clive"
then
      echo you are not Clive
fi
```

may be written as...

```
if [ ! "$name" = "Clive" ]
then
      echo you are not Clive
fi
```

The if else construct

Syntax:

```
if command1
then
      command2
      command3
      . . .
else
      command4
      command5
      . . .
fi
```

The if command executes *command1*. If the exit status of *command1* is zero, the commands between the then and the else are executed. If the exit status of *command1* is nonzero, the commands between the else and the fi are executed.

user2

```
if who | grep "^$1" > /dev/null
then
      echo "$1 is logged on"
else
      echo "$1 is not logged on"
fi
```

```
$ who
clive        tty8d         Jan  1 17:27
$ user2 clive
clive is logged on
$ user2 elaine
elaine is not logged on
$
```

The if else if construct

Syntax:

```
if command1
then
      command2
      . . .
else
      if command3
      then
            command4
            . . .
      fi
fi
```

If commands may be nested when more than just a two-way decision is

required.

```
if [ "$name" = "Colin" ]
then
      echo hello Daddy
else
      if [ "$name" = "June" ]
      then
            echo hello Mummy
      else
            echo hello
      fi
fi
```

The read command

Syntax:

```
read variables
```

The read command reads a line from the standard input and assigns the first word to the first variable in variables, the second word read to the second variable, and so on.

If there are more words on the line than variables listed, the excess words are assigned to the last variable.

readin

```
echo "? \c"
read a b
echo a
echo b
```

```
$ readin
? the rain in Spain
the
rain in Spain
$
```

readin2

```
echo "? \c"
read a
echo a
```

```
$ readin2
? the rain in Spain
the rain in Spain
$
```

The for command

Syntax:

```
for variable in thing1 thing2 ... thingn
do
        command1
        command2
        ...
done
```

The commands between 'do' and 'done' form the body of the loop. The 'for' command is executed; the first thing, *thing1*, is assigned to variable and the body of the loop is executed. Then, the second thing, *thing2*, is assigned to variable and the body of the loop is executed again. This cycle continues until there are no more things in the list.

forloop

```
for word in the rain in Spain
do
        echo $word
done
```

```
$ forloop
the
rain
in
Spain
$
```

The while and until commands

while syntax:

```
while command1
do
      command2
      command3
      ...
done
```

The commands between the 'do' and the 'done' are executed whilst

command1 returns an exit status of zero.

```
count=1
while [ "$count" -le 5 ]
do
      echo $count; count=`expr $count + 1`
done
```

until syntax:

```
until command1
do
      command2
      command3
...
done
```

The commands between the 'do' and the 'done' are executed until

command1 returns an exit status of zero.

```
count=1
until [ "$count" -gt 5 ]
do
      echo $count; count=`expr $count + 1`
done
```

The case command

Syntax:

```
case variable in
pattern1) commands;;
pattern2) commands;;
...
patternn) commands;;
esac
```

The contents of variable is successively compared with the patterns

pattern1, pattern2, ..., patternn until a match is found. When a match is found,

the commands after the bracket up to the double semicolon are executed.

chars

```
case "$1" in
[0-9]) echo digit;;
[a-z]) echo lowercase letter;;
[A-Z]) echo uppercase letter;;
esac
```

```
$ chars a
lowercase letter
$ chars +
$
```

The default case

The special character * matches anything, and is usually used at the end of a case statement to mark commands which will be executed if none of the other patterns are matched.

chars

```
case "$1" in
[0-9]) echo digit;;
[a-z]) echo lowercase letter;;
[A-Z]) echo uppercase letter;;
[*]) echo something else;;
esac
```

```
$ chars a
lowercase letter
$ chars +
something else
$
```

The exit command

The exit command enables the execution of a shell program to be terminated immediately.

Syntax:

```
exit number
```

Number is the exit status returned when the program is terminated.

Special echo characters

The following special characters may be used with the echo command.

\b	backspace
\c	suppresses newline
\f	form-feed
\n	newline
\r	carriage return
\t	tab character
\\	backslash character

Bibliography

Alphonce, Bo H. "Computer Applications: Analysis and Modelling." <u>Music Theory Spectrum</u> 11/1, pp.49–59, 1989.

_____. "Music Analysis by Computer—A Field for Theory Formation." <u>Computer Music Journal</u> 4,2, pp.26–35, 1980.

Babbitt, M. "The Use of Computers in Musicological Research." <u>Perspectives of New Music</u> 3, p.74, 1965.

Balaban, Mira. "Toward a Computerized Analytical Research of Tonal Music." Ph.D. diss., Rehovat, Israel, Weizmann Institute of Science, 386pp., 1981.

_____. "Towards a Computer Research of Tonal Music." <u>Proceedings of the Rochester 1983 International Computer Music Conference</u> comp. Robert W. Gross. San Francisco. Calif.: Computer Music Association, pp.138–60, 1984.

Bales, W., and Groom-Thornton, J. "AMUS: A Score Language for Computer-Assisted Applications in Music." <u>AEDS Proceedings of the 17th Annual Convention, Detroit, Michigan, May 14–18, 1979</u> Washington, D.C.:AEDS, pp.47–50, 1979.

Baroni, M. "A Project of a Grammar of Melody", <u>Proceedings of the Second International Symposium on Computers and Musicology</u> (Paris, CRNS, 1983), Orsay, pp.55–69, 1981.

Baroni, M., and Jacoboni, C. <u>Proposal for a Grammar of Melody: The Bach Chorales</u> Montreal, 1978.

Bauer-Mengelberg, S. "Music: The Photon Printer and DARMS." <u>Computers and the Humanities</u> 6, p.110, 1971–72.

Bent, I., and Morehen, J. "Computers in the Analysis of Music." <u>Proceedings of the Royal Musical Association</u> 104, pp.30–46, 1977–78.

Bernstein, L., and Olive, J. P. "Computers and the 16th-Century Chanson: A Pilot Project at the University of Chicago." <u>Computers and the Humanities</u> 3, pp.153–160, 1968–69.

Binkley, T. "Electronic Processing of Musical Materials." <u>Elektronische Datenverarbeitung in Der Musikwissenschaft</u> Harald Heckmann, ed., Regensburg: Gustav Bosse Verlag, pp.1–20, 1967.

Bowles, E. "Musicology and Computers." Computers and the Humanities 4, pp.207–219, 1969–70.

Brender, M. "Computer Transcription and Analysis of Mid-Thirteenth Century Musical Notation." Journal of Music Theory Yale School of Music Publication, 11,2, pp.198–221, 1967.

Brinkman, A. "A Binomial Representation of Pitch for Computer Processing of Musical Data." Music Theory Spectrum 8, pp.44–57, 1986.

_____. "A Data Structure for Computer Analysis of Musical Scores." Proceedings of the International Computer Music Conference 1984

_____. "A Design for a Single Pass Scanner for the DARMS Music Coding Language." Proceedings of the Rochester 1983 International Computer Music Conference comp. Robert W. Gross. San Francisco, Calif.: Computer Music Association, pp.7–30, 1984.

_____. "Toward a Library of Utility Computer Programs for the Music Theorist." Rochester, N.Y., Computer printout, Sept. 1975.

_____. Pascal Programming for Music Research Chicago: The University of Chicago Press, 1990.

_____. "Representing Musical Scores for Computer Analysis." Journal of Music Theory 30,2, 1986.

Camilleri, L. Carreras, F., Grossi, P. and Nencini, G. "A Software Tool for Music Analysis." Interface 16, pp.23–38, 1987.

Cantor, D. "A Computer Program that Accepts Common Musical Notation." Computers and the Humanities 6, pp.103–109, 1971–72.

Cobin, M. "Musicology and the Computer in New Orleans." Computers and the Humanities 1, pp.131–133, 1966–67.

Collins, W. S. "A New Tool for Musicology." Music and Letters XLVI, pp.122–125, 1965.

Cook, N. "Music Theory and 'Good Comparison': A Viennese Perspective." Journal of Music Theory 33,1, 1989.

Crane, F., and Fiehler, J., "Numerical Methods of Comparing Musical Styles." The Computer and Music Harry Lincoln, ed., Cornell University Press, pp.209–222, 1970.

Ellis, M. "Are Traditional Statistical Methods Valid for Quantitative Musical Analysis?" <u>Proceedings of the Second International Symposium on Computers and Musicology</u> (Paris, CRNS, 1983), Orsay, pp.185–195, 1981.

_____. "Linear Aspects of J. S. Bach's the Well-Tempered Clavier: A Quantative Survey." Diss., University of Nottingham, 1980.

Erickson, R. "A General-purpose System for Computer Aided Musical Studies." <u>Journal of Music Theory</u> 13,2, pp.276–294, 1969.

_____. "DARMS, A Reference Manual." New York: Queens College, CUNY, 1976.

_____. "Music Analysis and the Computer." <u>Journal of Music Theory</u> 12,2, pp.240–263, 1968.

_____. "Music Analysis and the Computer: A Report on Some Current Approaches and the Outlook for the Future." <u>Computers and the Humanities</u> 3, pp.87–104, 1968–69.

_____. "MUSICOMP 76 and the State of DARMS." <u>College Music Symposium</u> 17,1, pp.90–101, 1970.

Forte, A. "A Program for the Analytic Reading of Scores." <u>Journal of Music Theory</u> 10,2, pp.330–364, 1966.

_____. "Music and Computing: The Present Situation." <u>Computers and the Humanities</u> 2, pp.32–34, 1967–68.

Foxley, E. "The Harmonisation of Melodies for the Measurement of Melodic Variations." <u>Proceedings of the Second International Symposium on Computers and Musicology</u> (Paris, CRNS, 1983), Orsay, pp.93–114, 1981.

Gabura, A. "Music Style Analysis by Computer." <u>The Computer and Music</u> Harry Lincoln, ed., Cornell University Press, pp.223–276, 1970.

Gould, M. "A Keypunchable Notation for the Liber Usualis." <u>Elektronische Datenverarbeitung in Der Musikwissenschaft</u> Harald Heckmann, ed., Regensburg: Gustav Bosse, pp.25–40, 1967.

_____. "An Introduction to Computing in Music." <u>Computers and the Humanities</u> 2, pp.262–263, 1967–68.

Hewlett, W. and Selfridge-Field, E. "Computing in Musicology, 1966–91." <u>Computers and the Humanities</u> 25, pp.381–392, 1991.

Hiller, L., Jr. "Computer Music." Scientific American 201, pp.109–120, 1959.

_____. "Electronic and Computer Music." St. Louis Post-Dispatch Music
Section, May 17, 1959.

_____. and Isaacson, L. M. Experimental Music: Composition with an
Electronic Computer McGraw-Hill Book Co., New York, 1959.

_____. "Music composed with computers—a historical survey." The
Computer and Music Harry Lincoln, ed., Ithaca: Cornell
University Press, pp.42–96, 1970.

_____. "Some Structural Principles of Computer Music." Journal of the
American Musicological Society 9, pp.247–248, 1959.

_____, and Baker, R. "'Computer Cantata': A Study in Compositional
Method." Perspectives of New Music 3, p.62, 1965.

Holtzmann, S. "A Program for Key Determination." Interface 6, pp.29–56, 1977.

Howe, H. S., Jr. "Electronic Music and Computers." Perspectives of New Music
16, pp.70–84, 1977.

_____. "Some Combinational Properties of Pitch Structures." Perspectives
of New Music 4, pp.45–61, 1965.

Jackendoff, R. and Lerdahl, F. "Discovery Procedures vs. Rules of Musical
Grammar in a Generative Music Theory." Perspectives of New
Music 18, pp.503–510, 1979.

Kasemets, U. "Report from Ann Arbor." Music Quarterly 50, p.518, 1964.

Kassler, M. "Toward Music Information Retrieval." Perspectives of New Music
4, pp.59–67, 1966.

Klein, M. "Uncommon Uses for Common Digital Computers." Instruments and
Automation 30, pp.251–253, 1957.

Kostka, S. "Recent Developments in Computer-Assisted Musical Scholarship."
Computers and the Humanities 6, pp.15–21, 1971–72.

Kowalski, D. "An Algorithm and a Computer Program for the Construction of
Self-Deriving Arrays." In Theory Only 9,5–6, pp.27–50, 1987.

Landy, L. "Arabic Taqsim Improvisation: A Methodological Musical Study Using
Computers." Informatique et musique, Second Symposium
International ed. Hélène Charnassé. Ivry:ELMERATTO, CNRS,
pp.21–30, 1983.

_____. "Computer Musicology and Politics. Why are they never associated?" Proceedings of the Second International Symposium on Computers and Musicology (Paris, CRNS, 1983), Orsay, pp.243–253, 1981.

LaRue, J. and Cobin, M. W. "The Ruge-Seignelay Catalogue: An Exercise in Automated Entries." Elektronische Datenverarbeitung in Der Musikwissenschaft Harald Heckmann, ed., Regensburg: Gustav Bosse, pp. 41–56, 1967.

Lincoln, H. "Preliminary Studies of Melody as Wave Form." Proceedings of the Second International Symposium on Computers and Musicology (Paris, CNRS, 1983), Orsay, pp.43–54, 1981.

_____. "Some Criteria and Techniques for Developing Computerised Thematic Indices." Elektronische Datenverarbeitung in Der Musikwissenschaft, Harald Heckmann, ed., Regensburg: Gustav Bosse, pp.57–62, 1967.

_____. "The Current State of Music Research and the Computer." Computers and the Humanities 5, pp.29–36, 1970–71.

_____. "The Thematic Index: A Computer Application to Musicology." Computers and the Humanities 2, pp.215–220, 1967–68.

Logemann, G. "The Canons in the Musical Offering of J. S. Bach: An Example of Computational Musicology." Elektronische Datenverarbeitung in Der Musikwissenschaft Harald Heckmann, ed., Regensberg: Gustav Bosse, pp.63–88, 1967.

Martin, D. "Current Publications on Acoustics." Journal of the Acoustical Society of America 32, p.617, 1960.

Mason, R. Modern Methods of Music Analysis Using Computers Peterborough, N.H.: School House Press, pp.299, 1985.

Mathews. M. and Pierce, J. eds. Current Directions in Computer Music Research Mass.: MIT Press, 1989.

McLean, B. "The Design of a Portable Translator for DARMS." Proceedings of the 1980 International Computer Music Conference comp. Hubert S. Howe. San Francisco, Calif.: Computer Music Association, pp.246–64, 1982.

275

_____. "The Representation of Musical Scores as Data for Applications in Musical Computing." Doctoral dissertation, State University of New York at Binghamton, 1988.

Mendel, A. "Some Preliminary Attempts at Computer-Assisted Style Analysis in Music." Computers and the Humanities 4, pp.41–52, 1969–70.

Mongeau, M. and Sankoff, D. "Comparison of Musical Sequences." Computers and the Humanities 24, pp.161–175, 1990.

Morehen, J. "Statistics in the Analysis of Musical Style." Proceedings of the Second International Symposium on Computers and Musicology (Paris, CNRS, 1983), Orsay, pp.169–183, 1981.

Olson, H. F. and Belar, H. "Aid to Music Composition Employing a Random Probability System." Journal of the Acoustical Society of America 33,9, pp.1163–1170, 1961.

O'Maidin, D. "Computer Analysis of Irish and Scottish Jigs." Musical Grammars and Computer Analysis Atti Del Convegno, Pubblicati sotto gli auspici delle Università degli studi di Bologna e Modena, [Modena, 4–6 Ottobre 1982].

Page, S. "Computer Tools for Music Information Retrieval." D.Phil., Oxford University, 1988.

Pearce, A. "Computer Programs for Music Analysts." Ph.D., King's College, London

_____. "Troubadours and Transposition." Proceedings of the Second International Symposium on Computers and Musicology (Paris, CNRS, 1983), Orsay, p.167, 1981.

Pinkerton, R. C. "Information Theory and Melody." Scientific American 194, pp.77–86, 1956.

Plenckers, L. "A Pattern Recognition System in the study of the 'Cantigas de Santa Maria'." Musical Grammars and Computer Analysis Atti Del Convegno, Pubblicati sotto gli auspici delle Università degli studi di Bologna e Modena, [Modena, 4–6 Ottobre 1982].

Pruett, J. "The Harpur College Music-Computer Seminar: A Report." Computers and the Humanities 1, pp.34–38, 1966–67.

Rahn, J. "Toward a Theory for Chord Progression." Proceedings of the Second International Symposium on Computers and Musicology (Paris, CRNS, 1983), Orsay, pp.79–92, 1981.

Richardson, E. "Music and Electronic Computers." Nature 184, p.1754, 1959.

Rothgeb, J. "Musical Research by Computer: Some Current Limitations." Computers and the Humanities 5, pp.178–82, 1970–71.

_____. "Some Early Efforts in Computational Musicology." Computers and the Humanities 6, pp.56–58, 1971–72.

Russell, R. "A Set of Microcomputer Programs to Aid in the Analysis of Atonal Music." D.M.A., University of Oregon, 1983.

Schillinger, J. The Mathematical Basis of The Arts Reprint ed. (New York and London: Johnson Reprint, 1966).

Selleck, J. and Bakeman, R. "Procedures for the Analysis of Form: Two Computer Applications." Journal of Music Theory 9,2, pp.281–293, 1965.

Shannon, C. E. and Weaver, W. The Mathematical Theory of Communication The University of Illinois Press, 1949.

Smoliar, S. "Lewin's Model of Musical Perception Reflected by Artifical Intelligence." Computers in Music Research 2, pp.1–39, 1990.

_____. "Music Programs: An Approach to Music Theory through Computational Linguistics." Journal of Music Theory 20, pp.105–131, 1976.

Snell, J. "Computerized Hierarchical Generation of Tonal Compositions." Proceedings of the Second International Symposium on Computers and Musicology (Paris, CRNS, 1983), Orsay, p.197 1981.

Stech, D. "A Computer-Assisted Approach to Micro-Analysis of Melodic Lines." Computers and the Humanities 15, pp.211–21, 1981.

Suchoff, B. "Computerized Folk Song Research and the Problem of Variants." Computers and the Humanities 2, pp.155–158, 1967–68.

Vercoe, B. "Music Computation Conference: A Report and Commentary." Perspectives of New Music 13, pp.234–238, 1974.

Winograd, T. "Linguistics and the Computer Analysis of Tonal Harmony." Journal of Music Theory 12,1, pp.2–49, 1968.

Wittlich, G. "Computer Applications: Pedagogy." Music Theory Spectrum 11,1, pp.60–65, 1989.

_____., Schaffer, J. and Babb, L. <u>Microcomputers and Music</u> Englewood Cliffs., N.J.: Prentice-Hall, pp.321, 1986.