APL APPLIED IN MUSIC THEORY

Michael Kassler
Michael Kassler and Associates Pty Limited
Suite 2, 2 West Crescent Street
McMahons Point NSW 2060, Australia

1. Introduction. A formalised theory of a particular class of music is--or would be--a logical system that allows algorithmic determination whether any given musical composition belongs to the class and, if so, provides an automatic analysis of the composition's structure.

There is today no formalised theory of tonality, the musical 'language' that has governed most Western music composed between 1650 and 1900 (e.g., music by Bach or Brahms) and much subsequent popular music. However, a number of respected music theorists have proposed informally how theories of tonality could be formulated. I have been concerned for a number of years to explicate two different informal theories of tonality proposed by A. F. C. Kollmann (1756-1829) and Heinrich Schenker (1868-1935) respectively.

The polyphonic nature of Western music is well represented by matrices whose rows designate individual monophonic parts and whose columns represent simultaneities. This representation makes APL a preferred programming language for the development of functions whose arguments represent musical compositions and whose values are truth or falsehood according as particular structural relationships do or do not subsist amongst the arguments.

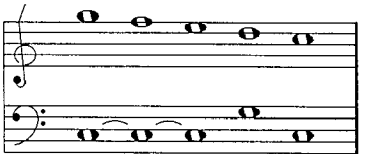
2. The Representation. Both theories can be explicated according to the principle of enharmonic equivalence, which holds that there are exactly twelve pitches in each octave, so that (for instance) C# and Db in the same octave have identical representation. This principle is adopted in the design of pianos and other keyboard instruments for which many musical compositions which are instances of tonality have been written. Any theoretically important differences between a C# and its enharmonically equivalent Db therefore should be calculable from this representation.

A pitch is represented by a *note code* calculated as $100x + y$, where x is the octave number between 1 and 9 inclusive and y is the pitch-class number between 0 and 11 inclusive. By convention, the octave from Middle-C to the next higher B is designated the fourth octave, and the pitch-class comprising all C's is designated 0, that comprising all C#'s is designated 1, etc. Accordingly, Middle-C itself is represented as 400 and the pitches immediately above and below Middle-C by 401 and 311 respectively.

Besides the note codes, two further *primitive units* are utilised: 8200, which designates a rest, and 8100, which indicates that the rightmost preceding note has been tied. The matrix

```
507  505  504  502  500
300 8100 8100  307  300
```

thus is a representation of the musical composition



Example 1

Whilst some primitive APL functions are immediately applicable to this representation of musical data, for the most part functions that carry out music-theoretically useful processes have had to be defined.

For instance, a dyadic function *TRANSPTO* has been defined whose value is the result of 'musically transposing' the left argument so that the right argument of this function becomes the rightmost note code of the result, provided that certain well-formedness criteria are satisfied; otherwise the value of *TRANSPTO* is 0.

If *X* is the matrix given above, the value of *X TRANSPTO* 507 is the matrix

```
602  600  511  509  507
307 8100 8100  402  307
```

which represents a musical composition that is

a transposition of *Example 1* seven semitones higher.

It will be seen that even this elementary function (from a music-theoretical perspective) departs from conventional addition in a number of ways. The $[1;2]^{th}$ element of *X* *TRANSPTO* 507 represents a pitch seven semitones above *X*[1;2], but to obtain 600 by 'adding' 7 to 505 requires functions that implement sterling arithmetic, i.e., 5/5 + 7*d* = 6/0, as the number of pitches per octave equals the number of old pence per shilling. And of course the primitive unit 8100, representing a tie, has to be unaltered by this 'addition'.

Of course, not every well-formed musical composition is an instance of tonality. One important criterion that compositions which are instances of tonality must meet is that they satisfy certain rules of counterpoint. These rules, amongst other things, identify certain combinations or successions of pitches as 'dissonant', restrict the use of dissonances in specific ways, and require that dissonances which are used appear in a context in which, in music-theoretical terminology, the dissonances are suitably prepared and resolved.

Our explications of the two theories of tonality include various APL functions which explicate appropriate counterpoint rules. These monadic functions have the general form *CP X*. They are propositional functions which have the value 1, representing truth, if the musical composition represented by the matrix *X* satisfies the particular counterpoint rules embodied in the function *CP*; otherwise, the value is 0, representing falsehood.

1.3 The Logical Structure of the Explications. Each of the two theories of tonality with which we are concerned claims that from certain elemental musical structures postulated as axioms other, more complex structures are derivable by successive application of rules of inference.

For example, in Heinrich Schenker's theory of tonality, *Example 1* above is an axiom (he called it, in German, an *Ursatz*). Schenker said or at least implied that, by successive application of rules of inference (called 'prolongation techniques'), entire movements of Beethoven symphonies and other masterworks of tonality could be derived from such axioms. However, music that was not an instance of tonality such as some compositions by Stravinsky could not be so derived.

Schenker and Kollmann illustrated their different theories of substantially the same music by means of numerous musical examples. Although both theories are plausible, neither author presented his theory with sufficient detail and rigour to allow a determination to be made whether it fully accounts for the music it purports to explain or, if not, whether it can be extended or modified so that it does account for this music. Indeed, before the availability of computers, it would have been practically impossible for the necessary tests to be undertaken.

Our work to explicate these theories is in progress. The general scheme is to explicate each of the two theories as a sequence of formalised languages L_1, L_2, ..., where the theorems of L_i usually become the axioms of L_{i+1}. These languages are all decidable, and a decision procedure, implementable in the form of an APL function, determines not merely whether a candidate musical composition represented by a matrix is a theorem of the language but if so provides automatically a derivation of the composition from an axiom.

Consider *Example 2*, shown on the next three pages. This output was produced by the monadic function *DERIVES1* when given as argument the matrix which represents the final musical composition in this example. *DERIVES1* implements a decision procedure for the formalised language S_1 which explicates an initial portion of Schenker's theory for major-mode compositions.

It will be seen that the axiom utilised in the proof of the composition is the same axiom shown earlier in *Example 1*. The various stages of the derivation illustrate particular prolongation techniques applicable to musical structures at the S_1 stratum.

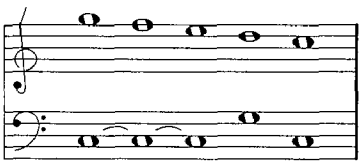
1.4 Some APL Functions. The workspace embodying the explication of S_1 contains some 1200 lines of APL code. Functions frequently invoke other functions which are logically more elementary. A full explanation would require a treatise rather than a short article.

By way of illustration, however, consider the function *ARTIC* displayed in *Example 3* which explicates Schenker's prolongation technique that we call in English 'articulation'. The third composition in *Example 2* has been derived from the second composition by means of this prolongation technique. Examination of the music reveals the salient feature of this technique--that with appropriate adjustment of notes on the lower of the two staves it has been possible to repeat all notes on the upper staff except the final note.

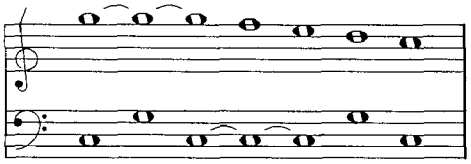
The function *ARTIC* begins with the assignment 0 (for falsehood) to its value. Only if all conditions are satisfied for the composition represented by the matrix *Y* to be derivable from the composition represented by the matrix *X* does the value of the function become 1 (for truth).

Lines 2 and 3 of this function, for instance, test whether *Y* and *X* satisfy particular counterpoint rules embodied in the APL function *CP2*. Unless both *Y* and *X* satisfy these rules, execution of *ARTIC* will terminate with value 0.

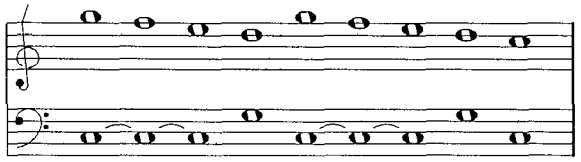
D E R I V A T I O N (IN THE SYSTEM S_1):



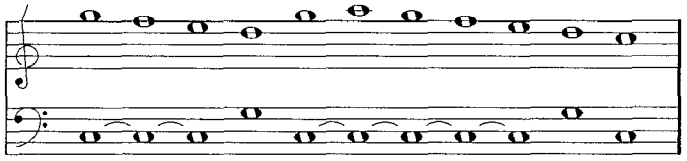
(5-TO-1 AXIOM.)



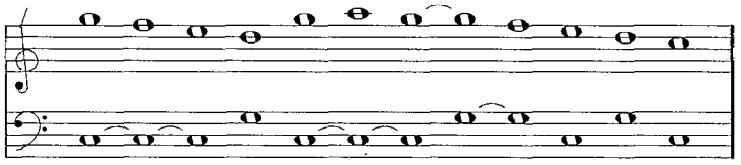
(INFERRED FROM LAST BY RULE OF BASS ARPEGGIATION.)



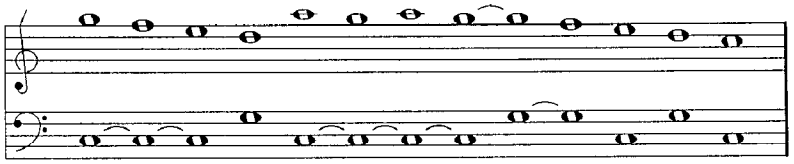
(INFERRED FROM LAST BY RULE OF ARTICULATION.)



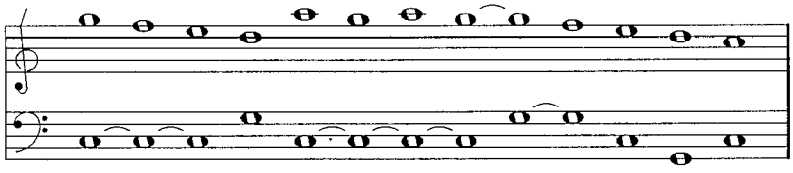
(INFERRED FROM LAST BY RULE OF NEIGHBOUR-NOTE PROLONGATION.)



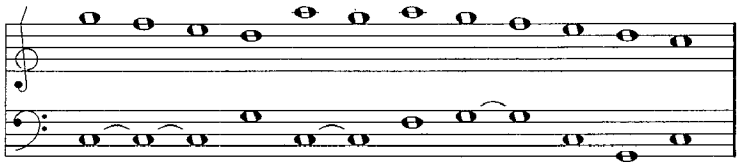
(INFERRED FROM LAST BY RULE OF BASS ARPEGGIATION.)



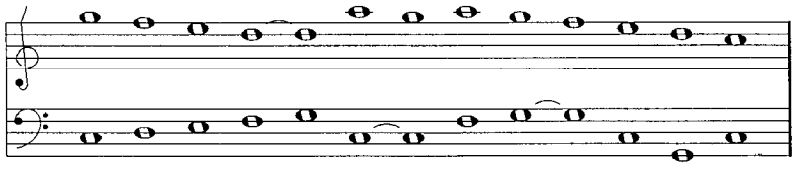
(INFERRED FROM LAST BY RULE OF NEIGHBOUR-NOTE PROLONGATION.)



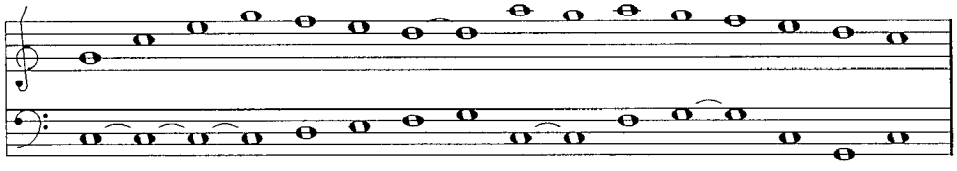
(INFERRED FROM LAST BY RULE OF BASS G TRANSFER.)



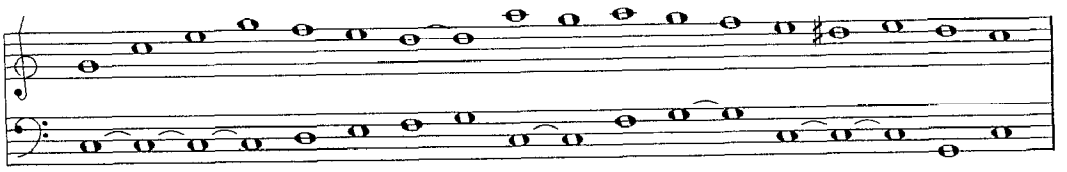
(INFERRED FROM LAST BY RULE OF BASS ASCENT.)



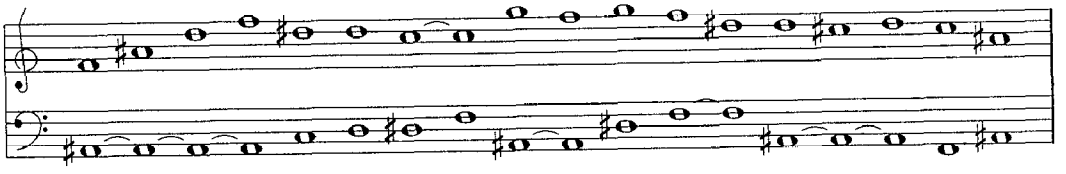
(INFERRED FROM LAST BY RULE OF BASS ASCENT.)



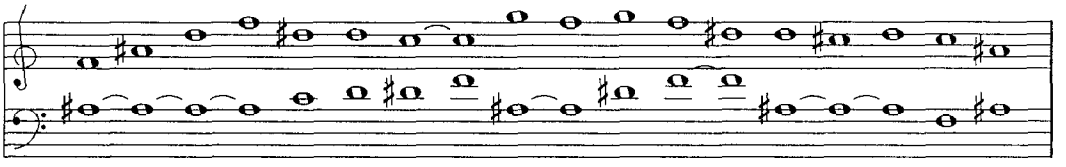
(INFERRED FROM LAST BY RULE OF PRELIMINARY ARPEGGIATION.)



(INFERRED FROM LAST BY RULE OF MIXTURE.)



(INFERRED FROM LAST BY RULE OF TRANSPOSITION TO A#4.)



(INFERRED FROM LAST BY RULE OF OCTAVE ADJUSTMENT OF LYNE 2 UP 1 OCTAVE.)

(Q. E. D.)

Example 2

The function *CP2* itself checks whether certain counterpoint rules are satisfied and calls on another function, *CP1*, for further counterpoint tests.

For *X ARTIC Y* to be true, *X* and *Y* must have identical endings, even if this be just the last musical simultaneity represented by the rightmost column of each respective matrix. This test is passed if execution of the *ARTIC* function proceeds to line 16 after line 15 rather than terminating. Line 22 tests whether the necessary repetition of notes on the upper staff, possibly with interspersed ties--this latter criterion is investigated by another function *TEXP*--has in fact been achieved.

1.5 Implementation. The explications are now implemented using STSC's APL*PLUS PC system, release 6.1, operating on a Canon A-200 personal computer system. This computer is compatible with the IBM PC/XT.

The output is printed on a Canon LBP-8 A2 laser beam printer. Musical notation is plotted by APL functions sending escape sequences which drive that printer's vector graphic mode. Plotting each note individually is slow. To prove that the final composition in *Example 2* is a theorem of S1 took 36.97 seconds; to print out *Example 2* took 34.57 minutes.

To avoid the slow speed of plotting music, I normally print out an alternative alphanumeric representation of musical notation. Faster musical typography would be obtainable by devising and downloading a suitable music font to the laser beam printer. The next generation of 80386-based personal computers of course will provide substantially faster processing.

Computer output of musical sound is useful, although the Canon A-200 by itself is limited to producing only one 'voice' at any one time.

1.6 References. The explication of Schenker's theory of tonality was begun in my PhD thesis *A Trinity of Essays* (Princeton

```
     ∇ VAL←X ARTIC Y;D;TOPL;BSC;ENDLN
[1]    VAL←0
[2]    →(~(CP2 Y))/0
[3]    →(~(CP2 X))/0
[4]    D←X[1;]⍳502
[5]   ⍝ FIND LEFTMOST D IN FIFTH OCTAVE TOP LYNE
[6]    →(D≥(¯1↑⍴X))/0
[7]    TOPL←JUSTNC(D↑X[1;])
[8]    →(4=⍴TOPL)/TOPL4
[9]    →(2≠⍴TOPL)/0
[10]   →(TOPL×.=(504 502))⍱(0,TOPLOK)
[11] TOPL4:→(~(TOPL×.=(507 505 504 502)))/0
[12] TOPLOK:→(~(X[1;D+1]∈(8100 8200)))/TEND
[13]   D←D+1 ◇ →TOPLOK
[14] TEND:ENDLN←D-(¯1↑⍴X)
[15]   →(~((,((2,ENDLN)↑X))×.=(,((2,ENDLN)↑Y))))/0
[16]   →(~((X[2;]LASTNC D)∈(207 307)))/0
[17]  ⍝ ENDS EQUAL
[18]   BSC←(300 300)SUBVIN ENDLN↓Y[2;]
[19]   →(BSC=0)/0
[20]   →(300∈(BSC[2]↓(ENDLN↓Y[2;])))/0
[21]   →(~((Y[2;]LASTNC(BSC[2]-1))∈(207 307)))/0
[22]   →(~((TOPL,TOPL)TEXP ENDLN↓Y[1;]))/0
[23]  ⍝ ENSURE THE REPETITION OF TOPL
[24]   →(~((ENDLN↓X[2;])TEXP ENDLN↓Y[2;]))/0
[25] HDNT:→((Y[1;]LASTNC BSC[2])=TOPL[1])/V1
[26]   →(Y[2;BSC[2]+1]≠8100)/0
[27]   BSC[2]←BSC[2]+1
[28]   →HDNT
[29] V1:VAL←1
     ∇
```

Example 3

University, 1967, available through University Microfilms Inc.). Here functions were systematically derived and presented in the mathematical form of primitive-recursive functions rather than in a programming language.

The transfer of these functions to APL was begun at the University of Sydney in 1974 with the support of a grant from the John Simon Guggenheim Memorial Foundation, and initial decision procedures were implemented then. After a pause of about ten years, this work is now being continued.

For information about the explication of A. F. C. Kollmann's theory of tonality which has been going on in parallel, see my article "Transferring a Tonality Theory to a Computer", *International Musicological Society, Report of the 12th Congress, Berkeley 1977* (Kassel, Bärenreiter, 1981), pp. 339-347. The decidability result mentioned is contained in my article "The Decidability of Languages that Assert Music", *Perspectives of New Music*, vol. 14-15 (1976), pp. 249-251.