Abstract:

The primary design objective of this system was to provide efficient and flexible tools for
the analysis of contemporary music, with a focus on set-theoretic and serial theory.

These software tools are useful in several con-
ctexts. In theoretical modeling, one is able to ex-
amine features of set classes or ordered pitch col-
lections, searching for specific properties. For
music analysis, the programs can extract structural
elements from the pitch material from serial compo-
sitions. A composer can use the programs to ex-
amine properties of typical compositional ma-
naterial in the form of either abstract set classes or
specific pitch classes.

The programs are written in the C programming
language and were implemented initially for the
UNIX operating system environment. Flexibility is
achieved through modular design, facilitating in-
tegration with other programs in the set, UNIX
utilites, and user-written programs. Most pro-
grams are command-driven, and the user controls
the flow of information between the database and
program modules. One can design one's own shell scripts that manipulate data from files, terminal input, or
other programs. While the programs are independent
as they stand, additions, features, and program
modules can be implemented and integrated with the

The Programs

The programs fall into four categories: library
procedures, main program, shell scripts, and
filters. The flow diagram above shows the rela-
tionships between these components and optional
user-written programs.

Set Class Library

User Programs (C. Kotzias, Marcel)

Main Programs

Shell, Commands

Set-Clas filters

Filters

The programs are divided into categories: set class
filters, main programs, shell scripts, and
programs. The flow diagram above shows the rela-
tionships between these components and optional
user-written programs.

Set Class Library

User Programs (C. Kotzias, Marcel)

Main Programs

Shell, Commands

Set-Clas filters

Filters

The programs are divided into categories: set class
filters, main programs, shell scripts, and
programs. The flow diagram above shows the rela-
tionships between these components and optional
user-written programs.
In order to facilitate the interchange between our program and the UNIX operating system, we have adopted a convention for differentiating set characteristics in the program output. These conventions are shown below:

1. Only brackets surround the listings of non-
   ordered pitch class elements.
2. Parameters surround the inversion vector.
3. Inversion vectors are used for the interval class vector.
4. Vortex brackets enclose the adjacent interval
   vectors.
5. The letter 'a' is used in front of the E-
   related set class. If no 1-set exists, a 'b'
   will follow.
6. The letter 'a' is used in front of the E-

We also use letters to identify the following:

```
Operands
prim     = primitive
inversion = inv
transposition = transp
altop        = altop
complement   = comp
```

```
Condition Relationships
if         = if
else       = else
then       = then
elseif     = elseif
endif      = endif
```

Using the Programs

The programs have been designed to be both flexible and easy to work with; with modest users and ex-

preferred programs will benefit from their use.

However, the user would be familiar with some fundam-
ental principles. We assume a basic knowledge of

UNIX utilities for file and data manipulation.

With this limited background, the design and usage

of the program should be easy to understand.

Only advanced advanced knowledge of the "pipes," small com-

mands, and expert capabilities of this kind, the

power of this system is extended greatly.

The output of commands and program output follows

the UNIX model. Programs are called by typing the

name of the program followed by arguments to the

program. The command is parsed according to the

program specifications, and the appropriate functions

are executed. The examples below illustrate usage:

The first example calls program format, parsing an

argument "26469", a single element of 12 pitch classes,

and prints the result:

```
# format 26469
```

which produces this output:

```
   (0 0 0 0 0 0 0 0 0 0 0 0)
```

in the second example several arguments are passed to

format in different formats. Each argument is

evaluating to determine if it is a set class name as

in the first two arguments, or a pitch class collec-

tion as in the third. The output of the program in-

cludes the same output as the arguments of the com-

mand line:

```
# format 26469 26468 0
```

which produces this output:

```
   (0 0 0 0 0 0 0 0 0 0 0 0)
```

some of the programs require specific argument struct-

ures, with each argument assigning different meanings

for that program. The program output, which shows us

a series of related pitch class elements, is followed by

an output representing a sub-

exclusivity for the duration of the related set class,

one of these programs. The following example demonstrates this principle:

```
# format 26469 26468 0
```

which produces this output:

```
   (0 0 0 0 0 0 0 0 0 0 0 0)
```

I conclude 08/02/44 3

*** Run: [08/02/44 3] ***

```
indications 3-note sets
```

```
Rotation 0: Subrows (054) set class 2-3-4 (013)
Rotation 1: Subrows (245) set class 3-4-1 (012)
Rotation 2: Subrows (452) set class 1-3-4 (014)
Rotation 3: Subrows (786) set class 1-4-3 (014)
Rotation 4: Subrows (245) set class 3-4-1 (012)
Rotation 5: Subrows (864) set class 1-4-3 (014)
Rotation 6: Subrows (542) set class 1-3-4 (014)
Rotation 7: Subrows (1452) set class 2-3-4-1 (018)
Rotation 8: Subrows (364) set class 1-4-3-2 (012)
Rotation 9: Subrows (4362) set class 3-2-1 (015)
```

ICMC '86 Proeceedings 332
The problem allows reports for specific information to be printed. The command line shown below requests only column 1, 2, and 3 of the above example: the net zone (implicit), the interval-class codes (1), and the prime time (e) for all interconnect line classes.

The general problem allows reports for specific information to be printed. The command line shown below requests only columns 1, 2, and 3 of the above example: the net zone (implicit), the interval-class codes (1), and the prime time (e) for all interconnect line classes.

1. genet -g 3
   The test results could be obtained by reformulating the command line as
   2. genet -v -o 3
   A more complex command line is illustrated by the "repeat-weather" option for genet. The option requests the input of a series of four characters, each of which represents information needed to satisfy the search routine. The command line appears as follows:
   3. genet -v -m 3
   The 'm' option for the artificial weather search requests the user, but the 'v' option signifies to the program that two sets of characters will represent the request to search for and complement map sets, the operator to view, the relation, and the frequency for use in the current search. In this case the program will examine all sets of interval codes and output only those sets that satisfy the specifications of mapping into the complement under inversion with a frequency greater than 6.

   4. genet -v -m 3
   The 'm' option for the artificial weather search requests the user, but the 'v' option signifies to the program that two sets of characters will represent the request to search for and complement map sets, the operator to view, the relation, and the frequency for use in the current search. In this case the program will examine all sets of interval codes and output only those sets that satisfy the specifications of mapping into the complement under inversion with a frequency greater than 6.
The program set is presented by three programs that are designed to be run interactively. Program 1 accepts a set of terms as a command-line argument and prints a matrix on the screen with transportation and order numbers. Then the user is prompted to enter a set of terms. Each set is entered, the locations of the set on the matrix are displayed in reverse video (highlight mode). The displayed set does not take order into account, and allow "striped" from the set at the end of each run. This program, which is helpful in analyzing or writing the time terms, uses the $I_{max}$, the object frame package and works for the number defined in 'mcent' that has highlight capabilities. The example shown below shows that the sink collection (SINK) occurs in the matrix as $I_{E}^{s}$ (period 7-6) and $I_{E}^{s}$ (0-4).

```
T 0 b 1 2 3 4 5 5 0 0 0 b 1 2 3 4 5 5 4 0 1 1 2 3 4 5 5 5 5 6 7 8 9 0 0 0 0 7 8 9 1 1 2 3 4 5 6 7 8 9 1 1 2 3

The user can obtain a summary of usage for each program by typing the program name on the command line with no arguments. The usage messages for each are shown below.
```

```python
def analyze_term(term):
    print(f"Input: {term}")
    # Process the term
```

```
Implementation
A central feature of the program is that net-class lists are obtained by inverting a set of structure data that are represented by matrix multiset. The program itself generates a matrix with an output of sets for the matrix that is shown below. This design reduces memory requirements, eliminates search time, and makes it possible to utilize binary operations for many processes, facilitating manipulation of a large set.

The set table, which is implemented as an array of structures, consists of two parts: the first contains the data for the individual set names and the second contains the location of the count of items. This table is handled as an input vector by symbol tables, with all sets in the same position in the same order. Each entry has a field that indicates the next set in the ordered list. Therefore, we can use binary search when generating sets, or linear search when looking up sets by name. In either case the length of the list is
```

```
Program 1 accepts a set of term names and prints a matrix showing the K and Kd relations among the group of terms (Franks, 1977).
```

```
```

ICMC 86 Proceedings

334
Shifting all bits to the left (with bit 12 rotating on position 0) transposes the set by one. We will refer to this operation as left-mutation:

\[
T(x) = \begin{cases} 
0 & \text{if } x \neq 0, \\
1 & \text{if } x = 0.
\end{cases}
\]

This is implemented as the C function rotate:

\[
\text{unsigned rotate(x) \{ x = x \ll 1; x = x \ll 12 \text{ if } x \neq 0; \} \} \}
\]

A function to perform this operation is shown below.

\[
\text{unsigned invert(x) \{ x = \neg x; return x; \} \}
\]

Biting is used to set arbitrary bits in a number. For example, to set bit 3 in set-taker s, we use a mask with bit 3 set to 1. The result of the operation s \& mask (inclusive OR) sets the bit.

\[
\begin{align*}
\text{mask:} & \quad 0100000001010101 \\
\text{S:} & \quad 0100000001010101 \\
\text{S \& mask:} & \quad 0100000001010101 \neq 0.
\end{align*}
\]

The bítline add operation is used to check for subset inclusion. For example, if we wish to see if set \( \{0,1,4\} \) occurs as a subset of another set \( \{0,1,2\} \), we can use a mask that represents \( \{0,1,4\} \). If \( \{0,1,2\} \) and \( \{0,1,4\} \) reside in the same set, then \( 0 \leftarrow 2 \) occurs in the set.

\[
\begin{align*}
\text{S:} & \quad 0100000001010101 \\
\text{mask:} & \quad 0100000000101010 \\
\text{S \& mask:} & \quad 0100000001010101 \neq 0.
\end{align*}
\]

336 ICMI-96 Proceedings
By repeating the operation after successive rotations of the mask, we can check for inclusion of each transformation of (111). The mask is then inverted and the process is repeated. In this manner we can quickly find all twelve-one operations under which one set maps onto another or into itself.

Most operations on pc sets are simplified by using techniques such as these. For example the prime form of a set is found by taking a set S and its inverse S'. Each is rotated 11 times and the form that yields the least value (i.e. a binary integer) represents the prime form.

Conclusions

The combined use of binary operations, in-memory storage of the database, and fast algorithms to search the database to optimize program efficiency, while independent program modules, careful design of program output, and the use of output filters increase utility and flexibility.

Using this program set, the user is able to combine zbine and C programs to analyze program patterns to use the information to find and access information in the database. In addition, the more experienced program developer can utilize the set-class library and any of the standard C libraries in the program's program design for more specific requirements.

When combined with a powerful shell programming language such as the C-shell, the ability to manipulate the information to manipulate data to output to further enhanced. The availability of job control, history of commands, and command line substitution all combine to allow the user to more easily program and to organize his general interests in the analysis of music, as well as the specific requirements of the moment.

References


Additional interval arrays are presented by Chirkin (1971). Since Chirkin follows Forte's normal order, we adjust the pattern for the six sets listed above.

The multiplicative operators N and P (sometimes called M and W) are discussed by Hume (1967) and by Forte. Under these operations each element of a pc set is multiplied by 0 or 7 (modulo 12). The primary characteristic of this transformation is that if 1 (as in (12)) maps into 5 (as in (5)) and vice versa.

The inversion vector (Morris, 1963) is an eight-position array assigned to the first four positions the number of times a set 5 maps into itself under the operations (12), (11), (10), (10), and in the last four positions the number of times a set maps into its complement under the next four operations.

The inversion matrix is discussed by Alphonse (1971).