Chapter 23

Rhythm

The subject of rhythm touches on nearly every aspect of music. Musical elements such as pitch, harmony, and dynamics can all be regarded from the point-of-view of temporal patterns of events. A number of complex tasks arise from rhythm-related In this chapter, two rhythm-related tools are introduced: **dur** and **metpos**.

The ***recip* Representation

For many types of processing tasks it is helpful to have a representation that encodes rhythmic information only. The `**recip` representation is simply a subset of `**kern` that excludes all information apart from the nominal note durations and common system barlines. In addition, `**recip` distinguishes rests from notes by including the 'r' signifier. Without an accompanying 'f↵r' a duration is assumed to pertain to a note.

Generating `**recip` data from `**kern` is straightforward using **humsed**. For a single-spine input, the following command will make the translation:

```
humsed '/^[^=]/s/[^0-9.r ]//g; s/^$/./' input.krn \
    | sed 's/\*\*kern/**recip/'
```

The first **humsed** substitution eliminates all data other than the numbers 0 to 9, the period, the lower-case `r`, and the space (for multiple-stops). Barlines remain untouched in the output. The second **humsed** substitution changes any empty lines to null data tokens; this might be necessary in the case of grace notes. The ensuing **sed** command is used simply to change the exclusive interpretation from `**kern` to `**recip`.

A simple type of processing might entail creating an inventory of rhythmic patterns. Suppose we wanted to determine the most common rhythmic pattern spanning a measure. Using a monophonic `**recip` input, we could use **context** to amalgamate the appropriate data tokens:

```
context -b ^= -o ^= input.recip | rid -GLId | sort \
    | uniq -c | sort -nr
```

The output for the combined voices of Bach's two-part Invention No. 5 shows just seven patterns. The most characteristic patterns are the second one: `8r 16 16 8 8 4 4` and the fourth one:

```
8 16 16 8 8 4 4.

   30   16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16
   12   8r 16 16 8 8 4 4
   11   8 8 8 8 16 16 16 16 8 8
    8   8 16 16 8 8 4 4
    1   8 8 8 8 2
    1   8 32 32 32 32 4 2
    1   4 4 16r 16 16 16 16 16 16 16
```

The *dur* Command

The **dur** command produces `**dur` output from either a `**kern` or `**recip` input. The `**dur` representation scheme consists simply of the elapsed duration of notes and rests, expressed in seconds. The following example shows a simple `**dur` representation (right spine) with a corresponding `**kern` input:

```
**kern    **dur
*         *MM60
=1        =1
12g       0.3333
12g       0.3333
12g       0.3333
4g        1.0000
4r        1.0000r
8g        0.5000
8g        0.5000
4g        1.0000
=2        =2
*-        *-
```

As in the case of `**recip`, the `**dur` representation designates rests via the lower-case `r` and uses the common system for barlines. Notice that `**dur` assumes a metronome indication of quarter-note equals 60 beats per minute if no other metronome marking is given.

Suppose that we wanted to estimate the total duration of some monophonic passage (ignoring rubato). We can do this by translating the score to `**dur`, eliminating everything but notes and rests, and sending the output to the **stats** command:

```
dur -d inputfile.krn | rid -GLId | grep -v '^=' | stats
```

The **-d** option for **dur** suppresses the outputting of duplicate durations arising from multiple-stops. Note that outputs from **dur** will adapt to any changes of metronome marking found in the input, so if the work accelerates the durations will be reduced proportionally.

The **-M** option will over-ride any metronome markings found in the input stream. For example, if we wanted to estimate the duration of a monophonic passage for a metronome marking of 72 quarter-notes per minute we could use the command:

```
dur -M 72 -d input.krn | rid -GLId | grep -v '^=' | stats
```

Of course, the duration of a passage is not the same as the length of time a given instrument sounds. Suppose, for example, that we wanted to compare the duration of trumpet activity in the final movements of Beethoven's symphonies. We need to make a distinction between the duration of notes and the duration of rests. Since the duration values for rests are distinguished by the trailing letter 'r', we can use **grep -v** to eliminate all rest tokens.

```
extract -i '*Itromp' inputfile.krn | dur -d | rid -GLId \
    | grep -v '^=' | grep -v r | stats
```

The **dur** command provides a **-e** option that allows the user to echo specified signifiers in the output. The **-e** option is followed by a regular expression indicating what patterns are to be passed to the output. This option allows us to "mark" notes of special interest. For example, suppose we wanted to determine the longest duration note for which Mozart had marked a staccato.

```
dur -e \' inputfile | rid -GLId | grep \' | sed 's/\'//' \
    | stats
```

The **-e** option ensures that `**kern` staccato marks (') are passed along to the output. The **rid** command eliminates everything but Humdrum data records. Then **grep** is used to isolate only those notes containing a staccato mark. The **sed** script is used to eliminate the apostrophe, and finally the numbers are passed to the **stats** command. The `max` value from **stats** will identify the duration (in seconds) of the longest note marked staccato.

This same basic pipeline can be used for a variety of similar problems. Suppose, for example, that we want to determine whether notes at the ends of phrases tend to be longer than notes at the beginnings of phrases — and if so, how much longer? In this case, we want to have **dur** echo phrase-related signifiers:

```
dur -e '{' inputfile | rid -GLId | grep '{' | sed 's/{//' \
    | stats
dur -e '}' inputfile | rid -GLId | grep '{' | sed 's/{//' \
    | stats
```

Similarly, do semitone trills tend to be shorter than whole-tone trills?

```
dur -e 't' inputfile | rid -GLId | grep 't' | sed 's/{//' \
    | stats
dur -e 'T' inputfile | rid -GLId | grep 'T' | sed 's/{//' \
    | stats
```

Of course, we can also use **dur** in conjunction with **yank** in order to investigate particular musical segments or passages. How much shorter is the recapitulation compared with the original exposition?

```
yank -s 'Exposition' -r 1 inputfile | dur | rid -GLId \
    | grep -v '=' | stats
```

```
yank -s 'Recapituation' -r 1 inputfile | dur | rid -GLId \
   | grep -v '=' | stats
```

Do initial phrases in Schubert's vocal works tend to be shorter than final phrases?

```
yank -m { -r 1 lied | dur | rid -GLId | grep -v ^= | stats
yank -m { -r $ lied | dur | rid -GLId | grep -v ^= | stats
```

How much longer is a passage if all the repeats are played?

```
thru inputfile | dur | rid -GLID | stats -o ^=
```

Recall that the **xdelta** command can be used to calculate numerical differences between successive values. If the input to **xdelta** is `**dur` duration information, then we can determine rates of change of duration. Most music exhibits lengthy passages of similar duration notes — as in a sequence of sixteenth notes. In French overtures, successive notes are often of highly contrasting durations (longer, very-short, long, etc.). Using **xdelta** we can identify such large changes of duration. For example, the following pipeline can be used to determine the magnitude of the *differences* between successive notes.

```
dur inputfile | xdelta -s ^= | rid -GLId | stats -o ^=
```

A small `mean` from **stats** will be indicative of works that tend to have smoother or less angular note-to-note rhythms.

Classifying Durations

We can use the **recode** command to classify durations into a finite set of categories. Suppose, for example, we wish to create a inventory of long/short rhythmic patterns. We might use **recode** with reassignments such as the following:

```
>=0.4   long
else    short
```

For a monophonic input, we can create an inventory of (say) 3-note long/short rhythmic patterns as follows:

```
dur inputfile | recode -f reassign -i '**dur' -s ^= | \
   context -n 3 -o = | rid -GLId | sort | uniq -c | sort -n
```

A typical output might appears as follows:

```
 230 long long long
3422 short short short
 114 long long short
 202 short short long
  38 long short long
 117 short long long
 194 long short short
```

```
114short long short
```

Notice that we might do a similar inventory based on durational *differences* rather than on durations. For example, the **xdelta** command will allow us to distinguish short*er* note relationships from long*er* relationships. Our reassignment file would be as follows:

```
==0   equal
>0    shorter
<0    longer
```

And our processing would be:

```
dur inputfile | xdelta -s ^= | recode -f reassign \
   -i '**Xdur' -s ^= | context -n 2 -o = \
   | rid -GLId | sort | uniq -c | sort -n
```

Using *yank* with the *timebase* Command

Recall that the **timebase** command can be used to reformat an input so that each data record represents an equivalent elapsed duration. For example, in a 4/4 meter, the following command will format the output so that each full measure consists of precisely 16 data records (not including the barline itself):

```
timebase -t 16 input.krn
```

Suppose we wanted to isolate all sonorities in a 4/4 work that occur only on the fourth beat of a measure. If we use **timebase**, we can ensure that the fourth beat always occurs a certain number of data records following the barline. For example, with the following command, the onset of the fourth beat will always occur 4 records follow the barline:

```
timebase -t 4 input.krn
```

We can now use **yank -m** to extract all appropriate sonorities. The "marker" is the barline and the "range" is 4 records following the marker, hence:

```
timebase -t 4 input.krn | yank -m ^= -r 4
```

Note that this process will extract only those notes that begin sounding with the onset of the fourth beat. Some notes may have begun prior to the fourth beat and yet are sustained into the beat. If we want to extract the *sounded* sonority, we can use the **ditto** command. Begin by expanding the work with a timebase that ensures all notes are present. For a work whose shortest note is a 32nd note, we can use an appropriately small timebase value. Then use the **ditto** command to propagate all sustained notes forward through the successive sonorities:

```
timebase -t 32 input.krn | ditto -s ^=
```

Now we can yank the data records that are of interest. Notice that the **-r** (range) option for **yank -m** allows us to select more than one record. This might allow us, say, to extract only

those sonorities that occur on off-beats. For example, the following command extracts all notes played by the horns during beats 2 and 4 in a 4/4 meter work:

```
extract -i '*Icor' input.krn | timebase -t 16 \
    | yank -m ^= -r 5-8,13-16
```

In some cases, we would like to yank materials that do not themselves contain explicit durational information. Suppose, for example, that for a waltz repertory, we want to contrast those chord functions that tend to occur on the first beat with those that happen on the third beat. We will need to have an input that includes both a `**harm` spine encoding the Roman numeral harmonic analysis, as well as one or more `**kern` or `**recip` spines that include the durational information. We can use the **timebase** command to expand the output accordingly — cuing on the duration information provided by `**kern` or `**recip`. Having suitable expanded the input, we can dispense with everything but the `**harm` spine. For works in 3/4 meter, the following pipeline would provide an inventory of chords occuring on the first beat of each bar:

```
timebase -t 8 input | extract -i '**harm' \
    | yank -m ^= -r 1 | rid -GLId | sort | uniq -c | sort -n
```

And the following variation would provide an inventory of chords occuring on the third beat of each bar. (There are 6 eighth durations in a bar of 3/4, therefore the beginning of the third beat will coincide with the 4th eighth — hence the range `-r 4:`)

```
timebase -t 8 input | extract -i '**harm' \
    | yank -m ^= -r 4 | rid -GLId | sort | uniq -c | sort -n
```

The *metpos* Command

The **metpos** command generates a `**metpos` output spine containing numbers that indicate the metric strength of each sonority. By "metric position" we mean the position of importance in the metric hierarchy for a measure.

The highest position in any given metric hierarchy is given by the value '1'. This value is assigned to the first event at the beginning of each measure. In duple and quadruple meters, the second level in the metric hierarchy occurs in the middle of the measure and is assigned the output value '2'. (In triple meters, **metpos** assumes that the second and third beats in the measure are both assigned to the second level in the metric hierarchy.) All other metric positions in the measure (beats, sub-beats, sub-sub-beats, etc.) are assigned successively increasing numerical values according to their placement in the metric hierarchy. In summary, larger `**metpos` values signify sonorities of *lesser* metric significance.

By way of illustration, consider the case of successive eighth notes in a 2/4 meter. The metric hierarchy values for successive eighths are: 1, 3, 2, 3. In the case of successive sixteenth notes in 2/4, the metric hierarchy values are: 1,4,3,4,2,4,3,4. In the case of 6/8 meter, successive sixteenth durations exhibit a metric hierarchy of: 1,4,3,4,3,4,2,4,3,4,3,4.

For correct operation, the **metpos** command must be supplied with an input that has been for-

matted using the **timebase** command. That is, each data record (ignoring barlines) must represent an equivalent duration of time. In addition, **metpos** must be informed of both the *meter signature* and the *timebase* for the given input passage. This information can be specified via the command line, however it is usually available in the input stream via appropriate tandem interpretations.

The following extract from Bartók's "Two-Part Study" No. 121 from *Mikrokosmos* demonstrates the effect of the **metpos** command. The two left-most columns show the original input; all three columns show the corresponding output from **metpos**:

```
**kern    **kern  **metpos
*tb8      *tb8    *tb8
=16       =16     =16
*M6/4     *M6/4   *M6/4
8Gn       8b-     1
8A        8ccn    4
8B-       8cc#}   3
8cn       {8f#    4
8c#}      8gn     3
{8F#      8a      4
8G        8b-     2
8A        8ccn    4
8B-       4b-     3
8cn       .       4
8c#}      8fn}    3
8r        8r      4
=17       =17     =17
*M4/4     *M4/4   *M4/4
8d        2r      1
4.d       .       4
.         .       3
.         .       4
{2d_      8dd     2
.         4.dd    4
.         .       3
.         .       4
=18       =18     =18
8d        {1dd_   1
8A        .       4
8F#       .       3
8E        .       4
8D        .       2
8BB       .       4
8D        .       3
8E}       .       4
=19       =19     =19
*M3/2     *M3/2   *M3/2
{8F#      8dd     1
```

```
8A      8ffn    4
8c#     8aa     3
8A      8ff     4
8F#     8dd     2
8A      8ff     4
8F#     8dd     3
8E      8ccn    4
8D      8b-     2
8BBn    8gn     4
8D      8b-     3
8E}     8cc     4
=20     =20     =20
*-      *-      *-
```

Notice that **metpos** adapts to changing meter signatures, and correctly distinguishes between metric accent patterns such as 6/4 (measure 16) and 3/2 (measure 19).

The `**metpos` values provide additional ways of addressing various rhythmic questions. We might use **recode** for example, to recode the numerical outputs from **metpos** into a smaller set of discrete categories. For example, we might classify metric positions using the following reassignment file:

```
==1   strong
>=3   secondary
else  weak
```

The words 'strong', 'secondary', and 'weak' can then be sought by **grep** or **yank -m** allowing us to isolate points of particular metric stress. Since **metpos** adapts to changing meters, we can confidentaly process inputs that may contain mixtures of meters.

Changes of Stress

Once again we can make use of **xdelta** to identify relationships between successive metric position values. Suppose we had a collection of Hungarian melodies and we wanted to determine how each degree is approached in terms of metric strength. That is, we would like to count the number of tonic pitches that are approached by a weak-to-strong context versus the number of tonic pitches approached by a strong-to-weak context. We also want similar measures for supertonic, mediant, subdominant, etc. scale degrees.

This task involves creating an inventory where fourteen different items are possible: (1) tonic strong-to-weak, (2) tonic weak-to-strong, (3) supertonic, strong-to-weak, etc. A suitable inventory will involve creating two spines of information — scale-degree and relative metric strength.

Assuming that our Hungarian melodies encode key information, creating a `**deg` spine is straightforward. Recall that the **-a** option for **deg** avoids distinguishing the direction of approach (from above or below):

```
deg -a magyar*.krn > magyar.deg
```

Creating a spine encoding relative metric strength will be more involved. First we need to expand our input according to the shortest note. We use **census -k** to determine the shortest duration, and then expand our input using **timebase**.

```
census -k magyar*.krn
timebase -t 16 magyar*.krn > magyar.tb
```

Using **metpos** will allow us to create a spine with the metric position data.

```
metpos magyar.tb > magyar.mp
```

Note that **metpos** automatically echoes the input along with the new `**metpos` spine. At this point, the result might look as follows:

```
!!!OTL: Graf Friedrich In Oesterraaich sin di Gassen sou enge
**kern          **metpos
*ICvox          *
*Ivox           *
*M3/4           *M3/4
*k[f#]          *
*G:             *
*tb16           *tb16
{8g             2
.               4
8b              3
.               4
=1              =1
8dd             1
.               4
etc.
```

We want to be able to say that the relationship between the first eighth-note G and the eighth-note B is "strong-to-weak" and that the relationship between the eighth-note B and the eighth-note D is "weak-to-strong." In order to procede we need to eliminate all of the data records that contain only a metpos value — that is, there is no pitch present in the `**kern` spine. We can do this using **humsed**; we simply delete all lines that begin with a period character:

```
humsed '/^\./d' magyar.mp
```

The result is as follows:

```
!!!OTL: Graf Friedrich In Oesterraaich sin di Gassen sou enge
**kern          **metpos
*ICvox          *
*Ivox           *
*M3/4           *M3/4
*k[f#]          *
```

```
*G:             *
*tb16           *tb16
{8g             2
8b              3
=1              =1
8dd             1
etc.
```

Notice that the successive `**metpos` values will now allow us to characterize the changes in stress between successive notes: 2 followed by 3 indicates a strong-to-weak change of metric position, 3 followed by 1 indicates a weak-to-strong change of metric position. We can use **xdelta** to calculate the differences in metric position values: positive differences will indicate weak-to-strong changes and negative differences will indicate strong-to-weak changes. If both values have the same metric position value, then the successive notes hold equal positions in the metric hierarchy. Before using **xdelta** we need to isolate the `**metpos` spine using **extract**:

```
humsed '/^\./d' magyar.mp | extract -i '**metpos' \
    | xdelta -s ^=
```

The result is:

```
!!!OTL: Graf Friedrich In Oesterraaich sin di Gassen sou enge
**Xmetpos
*
*
*M3/4
*
*
*tb16
.
1
=1
-2
etc.
```

Now we can use **recode** to classify the changes of metric position according. Our reassignment file (named `reassign`):

```
>0    strong-to-weak
<0    weak-to-strong
==0   equal
```

Appending the appropriate command:

```
humsed '/^\./d' magyar.mp | extract -i '**metpos' \
    | xdelta -s ^= | recode -f reassign -i '**Xmetpos'  -s
    ^= > magyar.xmp
```

Now we can assemble the resulting metric change spine with our original **deg spine. Each data record will contain the scale degree in the first spine and the change of metric position data in the second spine. The final task is to create an inventory using **rid**, **sort** and **uniq**:

```
assemble magyar.deg magyar.xmp | rid -GLId | grep -v ^= \
   | sort | uniq -c
```

The final result will appear as below. The first output line indicates that there were three instances of a tonic pitch approached by a note of equivalent position in the metric hierarchy. The second line indicates that there were twenty-five instances of a tonic pitch approached by a note having a stronger metric position:

```
 3   1   equal
25   1   strong-to-weak
30   1   weak-to-strong
 3   2   equal
14   2   strong-to-weak
13   2   weak-to-strong
 1   3   equal
39   3   strong-to-weak
34   3   weak-to-strong
 3   4   equal
26   4   strong-to-weak
17   4   weak-to-strong
13   5   equal
49   5   strong-to-weak
42   5   weak-to-strong
 1   6   equal
13   6   strong-to-weak
14   6   weak-to-strong
 3   7   strong-to-weak
 6   7   weak-to-strong
 1   7-  weak-to-strong
 3   r   equal
10   r   strong-to-weak
```

Instead of scale degree, any other Humdrum spine might be used. For example, if the input contained functional harmony data (**harm) then the output inventory would identify how particular chord functions tend to be approached. For example, we could establish whether the submediant chord is more likely to be approached in a strong-to-weak or weak-to-strong rhythmic context. Similarly, this same technique can be used to determine whether particular melodic or harmonic intervals tend to be approached using particular stress relationships.

In addition, our input spine might also be transformed via the **context** command. Given a **harm spine, for example, **context** could be used to generate two-chord harmonic progressions. This would permit us to determine, for example, whether a specific progression such as *ii-V* tends to fall in strong-to-weak or weak-to-strong contexts.

Reprise

There are a vast number of issues raised in rhythm-related processing. In this chapter we have touched on a few of the more basic tasks. These include identifying the durations of various passages using **dur**; classifying and contextualizing durations using **recode** and **context**; isolating particular rhythmic moments using **timebase** and **yank -m**; determining relative metric positions using **metpos**; and characterizing metric syncopation using **synco**.

Processing data that does not explicitly contain duration-related information (such as `**harm` or `**deg`) often requires some preparation. It is often useful to maintain a coordinated file where the spines of interest are linked with duration-related spines that assist in processing.

One further topic related to rhythm remains to be discussed. The **accent** command allows the user to distinguish notes according to their estimated perceptual importance. We will consider **accent** in Chapter 31.